Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Turšič

Izdelava grafičnega vmesnika za na argumentiranju temelječ inteligentni tutorski sistem

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Matej Guid

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Izdelava grafičnega vmesnika za na argumentiranju temelječ inteligentni tutorski sistem (angl. *Development of graphical user interface for argument-based intelligent tutoring system*)

Tematika naloge:

"Izdelajte grafični vmesnik za inteligentni tutorski sistem, ki temelji na argumentiranju oz. na argumentiranem strojnem učenju in je trenutno realiziran kot konzolna aplikacija v programskem jeziku Python."

Zahvaljujem se doc. dr. Mateju Guidu za pomoč, čas, motivacijo in koristne nasvete pri nastajanju diplomskega dela. Zahvalil bi se tudi dekletu ter vsem bližnjim, ki so vame verjeli, me spodbujali in podpirali izbrano pot. Hvala!

Where there's a will, there's a way.

Kazalo

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
ABML	argument-based machine laerning	argumentirano strojno učenje
ITS	intelligent tutoring system	inteligentni tutorski sistem
GUI	graphical user interface	grafični uporabniški vmesnik
UI	user interface	uporabniški vmesnik
UX	user experience	uporabniška izkušnja
UIX	user interaction and experience	uporabniška izkušnja in interakcija
API	application programming interface	aplikacijski vmesnik
HTML	hyper text markup language	označevalni jezik za spletne strani
CSS	cascading style sheets	slogovna predloga
PHP	hypertext preprocessor	strežniški programski jezik
XML	extensible markup language	razširljivi označevalni jezik
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
TCP	transmission control protocol	transportni kontrolni protokol
JSON	JavaScript object notation	preprost format za izmenjavo podatkov
RAM	random-access memory	bralno-pisalni pomnilnik
MVC	model-view-controller	arhitektura model-pogled-kontroler

Povzetek

Grafični vmesniki (angl. *graphical user interface, GUI*) v današnjem času predstavljajo pomemben del računalniških sistemov. Srečujemo jih tako rekoč povsod, pa naj bo to mobilni telefon, tablica, računalnik, delovni stroj, avtomobil, ura, nadzorne plošče v elektrarni in še bi lahko naštevali. Z množično uporabo svetovnega spleta se je povečala priljubljenost spletnih grafičnih vmesnikov. Tako lahko prek spleta uporabljamo najrazličnejše portale in aplikacije ter npr. celo nadzorujemo pametne hiše. S svojo preprostostjo v večini primerov omogočajo, da jih lahko brez posebnega znanja uporablja kdor koli.

Naloga diplomske naloge je izdelati grafični vmesnik za obstoječ in na argumentiranju temelječ inteligentni tutorski sistem (angl. *intelligent tutoring system, ITS*). Pri tem je bilo pomembno, da je grafični vmesnik skladen z načinom uporabe inteligentnega tutorskega sistema, ki je do zdaj deloval le kot konzolna aplikacija. ITS pri svojem delovanju uporablja veliko podatkov. Ob njihovi implementaciji v grafičnem vmesniku je pomembno, da jih pravilno strukturiramo, saj bodo le tako prišli do izraza. Prvi izziv nam je torej predstavljal prikazovanje podatkov sistema. Pri tem moramo imeti v mislih tudi to, da inteligentni sistem, ki teče v ozadju, podpira delo z najrazličnejšimi podatki iz različnih učnih domen. Drugi pomemben izziv je bil povezan s komunikacijo med inteligentnim tutorskim sistemom in spletnim strežnikom, na katerem je grafični vmesnik.

Pri reševanju izzivov smo uporabili več različnih metod. Za pomoč pri načrtovanju smo uporabili razna načela, ki jih najdemo v literaturi in na

spletu. Ustvarili smo preprost in pregleden grafični vmesnik, ki temelji na enostranskem prikazu strani. S tem omogočimo dinamičen prikaz in optimiziramo čas nalaganja prikaza. S primernim izborom barv naredimo privlačen videz vmesnika, pravilne postavitve elementov pa nam prinesejo dobro uporabniško izkušnjo. Grafični vmesnik smo povezali z uporabo vtičnice (angl. *socket*) in tako omogočili komunikacijo v realnem času. Razvita spletna aplikacija med drugim omogoča tudi to, da učitelj vnaprej določi napredne koncepte iz izbrane učne domene, na katere naj bo učenec še posebej osredinjen pri podajanju argumentov. Grafični vmesnik nato pri izbranih konceptih sproti prikazuje napredek učenca. Delovanje vmesnika smo prikazali na učni domeni, v kateri se učenec uči razumeti in utemeljevati bonitetne ocene podjetij.

Ključne besede: umetna inteligenca, izobraževanje, grafični vmesnik, inteligentni tutorski sistemi, argumentirano strojno učenje, interaktivna zanka za zajemanje znanja, argumentiranje, bonitetne ocene.

Abstract

Graphical user interface, GUI, today, constitutes an important part of a computer system. They can be found almost everywhere, whether in a cell phone, tablet, computer, machine, car, watch or control panel in a power plant, to name a few. The massive role of the internet has increased the popularity of web GUI's. Via the internet we can use a variety of online portals and applications, for e.g. even to control smart houses. Due to their simplicity they can be used by anyone, in most cases, without having special knowledge.

The task of this thesis was to create the graphical interface for an existing, reasoning-based intelligent tutoring system. It was important that the GUI was consistent with the mode of use of the intelligent tutoring system, ITS, which, until now, operated only as a console application. ITS uses a large amount of data in its operation, therefore, upon implementation of these in the graphical user interface, it is most important that they are properly structured, as only then can they come to the fore. The first challenge was therefore the data display system. Here, we had to bear in mind that the intelligence system running in the background also supported the work of different data for a variety of didactic domains. Another important challenge was related to communication between the intelligent tutoring system and the web server, on which the GUI is located.

In addressing these challenges we used several different methods. To assist in planning, we used various principles found both in the literature and on the web. We created a simple and clear graphical interface, based on a one-sided

page view. This enabled a dynamic display and optimised the display loading time. With an appropriate selection of colours, we contrived an attractive appearance for the interface and the correct setting up of elements, that all together brought a good user-experience. We connected the graphical interface using a socket, enabling real-time communication. The developed web application, among other things, allows the teacher to determine the advanced concepts in the selected didactic domain, upon which the student will be specially focused when explaining learning examples. The GUI, then gives a continual display of the progress of the student in these selected concepts. The operation of the graphical user interface was demonstrated in didactic domain, where students learn how to understand and establish company credit ratings.

Keywords: artificial intelligence, education, graphical user interface (GUI), intelligent tutoring system (ITS), argumentative machine learning, interactive loop for capturing knowledge, advocating, credit assessments.

Poglavje 1

Uvod

Živeti pomeni učiti se. Pomeni spoznavati, se razvijati, razumeti sebe in svet. Kar počnemo smo se naučili, želja po znanju pa nas žene naprej. Da bi temu ostalo tako, potrebujemo dobre učitelje. Ameriški psiholog Benjamin Bloom je že leta 1984 s svojo študijo[1] nakazal, da bi se proces učenja lahko izboljšal s pristopom ena na ena [2]. Kot del definicije dobrega učitelja bi lahko navedli, da ima vsak učenec svojega učitelja, saj se s tem izboljša uspešnost učenca. Sprva se je takšna možnost zdela neizvedljiva, kasneje pa se je pojavila ideja o realizaciji z uporabo inteligentnih tutorskih sistemov (angl. *intelligent tutoring systems*; v nadaljevanju: ITS).

Čeprav dandanes učenje večinoma poteka v razredih, je znano, da je poučevanje učenca z osebnim učiteljem veliko bolj uspešno, vendar cenovno večkrat nedosegljivo. Ocenjeno je bilo, da se uspešnost s takšnim pristopom izboljša tudi za dva standardna odklona [2]. Na podlagi teh dognanj pa obstaja veliko zanimanje za sisteme računalniškega podprtega poučevanja (angl. *computer-aided/computer-assisted instruction, CAI*). Uporaba CAI je lahko bolj učinkovita od poučevanja v razredu. Njihova pomanjkljivost je le v statičnem, togem obnašanju, saj se niso zmožni prilagoditi specifičnim potrebam učencev.

Bolj ambiciozen pristop k reševanju tega problema nam daje uporaba

[1] Znana kot "Two-sigma problem".

inteligentnih tutorskih sistemov (ITS), ki omogočajo interakcijo z učencem in se prilagajajo njegovim potrebam. ITS so računalniški sistemi, pri katerih govorimo o interdisciplinarnem področju, ki povezuje pedagoške vede, računalništvo (umetna inteligenca) in psihologijo (kognitivna znanost) [24]. Namenjeni so poučevanju in so sestavljeni iz več modelov. Vsebujejo model domene, pedagoški model (model učitelja) in model učenca. Zaradi sprotne gradnje in ažuriranja modela učenca so se sposobni prilagajati specifičnim potrebam vsakega učenca posebej. Model nam pove kaj učiti in s pomočjo strategije poučevanja tudi kako učiti. Tako lahko omogočamo, da ima učenec večjo kontrolo nad svojim učenjem in ima popoln nadzor nad reševanjem raznih učnih primerov. Z vsakim rešenim primerom mu sistem vrne odziv in sporoči kakšen rezultat je dosegel, hkrati mu lahko poda tudi napotke in namige, kako priti do boljše rešitve ali do enakega rezultata po krajši poti.

V primerjavi s CAI sistemi, lahko pri ITS dosežemo bistveno boljšo učinkovitost. Groba ocena trenutnih izvedb se nahaja nekje na polovici poti med učenjem v razredu ter učenjem z osebnim učiteljem. Primer uspešnega ITS je Carnegie Learning,[2] ki ga v ZDA za učenje matematike uspešno uporablja več kot 650.000 učencev in dijakov.

V tem delu nadaljujemo razvoj ideje, da bi uporaba ITS sistemov v kombinaciji s tehnologijo argumentiranega strojnega učenja (angl. *argument-based machine learning, ABML*) [15] in interaktivne zanke za zajemanje znanja (angl. *knowledge refinement loop*), lahko omogočila razvoj nove vrste ITS: na argumentiranju temelječe inteligentne tutorske sisteme [27].

ABML združuje tehniko strojnega učenja s tehnikami argumentiranja. Z argumenti domenski ekspert razlaga odnose med značilkami oz. atributi, ki jih postavlja nad danimi kritičnimi primeri. Pri tem se mu ob izbiri atributa prikazujejo protiprimeri in mu lajšajo delo argumentiranja. Interakcija poteka s pomočjo interaktivne zanke za zajemanje znanja, skupaj z ABML pa nam predstavlja zmogljivo orodje tako za zajemanje ekspertnega znanja kot tudi za interaktivni učni proces z učencem. Takšno orodje je možno uporabiti

[2] `http://www.carnegielearning.com`

za na argumentiranju temelječ tutorski sistem.

Ideja o na argumentiranju temelječih inteligentnih sistemih je bila predstavljena v članku "Designing an interactive teaching tool with abml knowledge refinement loop" [27]. Nadaljevana je bila v magistrskem delu Matevža Pavliča [17], kjer je bil sistem povezan v celoto in nadgrajen z dodano interakcijo preko konzole, razvite pa so bile tudi mere za ocenjevanje podanih argumentov. Nastalo aplikacijo je tako moč upravljati preko konzole, kar sicer ni najbolj praktično za osebe, ki niso vešče uporabe le-te. Glede vizualnega prikaza je terminal ne le manj priročen (npr. ker uporablja le črno-bele elemente), pač pa tudi bistveno manj pregleden. Nastala je ideja o nadgradnji in uporabi pravega grafičnega prikaza. Z uporabo grafičnih vmesnikov se pojavi tudi merilo uporabniške izkušnje, ki ga določa kakovost interakcije. Za uspešnejšo interakcijo je torej pomembna boljša uporabniška izkušnja.

Namen tega diplomskega dela je izdelava grafičnega vmesnika za na argumentiranju temelječ tutorski sistem, s katerim se uporaba ITS-a približa širši populaciji. S tem se tudi izboljša kakovost komunikacije človek - računalnik. Tako bi lahko tako rekoč vsak uporabljal takšne sisteme in imel svojega "učitelja". Izziv pri realizaciji se najprej pojavi pri prikazu velikega števila podatkov na majhni površini, uporabi pravih oblik elementov in izbiri primernih barv vmesnika. Drugi izziv pa je, kako komunicirati med obstoječim algoritmom ITS in novim sistemom grafičnega vmesnika, ki je napisan v drugem programskem jeziku. Pri izdelavi je pomemben podatek tudi ta, da celoten sistem deluje neodvisno od domene. To pomeni, da lahko kot uporabniki izberemo poljuben nabor podatkov, sistem pa nas z uporabo le-teh tudi poučuje.

V delu najprej opišemo metode razvijanja grafičnih vmesnikov in analize podobnih sistemov. Dotaknemo se teme uporabniške izkušnje in načrtovanja grafičnega vmesnika, kjer so opisani postopki in načela, ki so pri razvoju v pomoč. Za tem je na podlagi primera opisan postopek argumentiranega strojnega učenja in interaktivne zanke za zajemanje znanja. Opišemo tudi komunikacijo obstoječe aplikacije s sistemom grafičnega vmesnika. Tretje

poglavje zajema rezultate in pojasni princip delovanja učne seje v grafičnem vmesniku. Poleg tega je opisana tudi krajša raziskava o primernosti izbiranega grafičnega vmesnika, glede na namen uporabe. Za pridobivanje podatkov pa smo pri uporabi anketirali testne uporabnike grafičnega vmesnika. Četrto poglavje zajema opis izzivov, ki so se pojavili pri realizaciji sistema. Delo zaključimo v petem poglavju, kjer so zapisane sklepne ugotovitve.

Poglavje 2

Metode

V tem poglavju bodo opisane metode pri razvoju izdelka. Od podrobnosti pri načrtovanju grafičnega vmesnika do preoblikovanja algoritma v strežniški servis (angl. *service*), ki na strežniku služi kot aplikacijski vmesnik (v nadaljevanju API) za komunikacijo s spletnim strežnikom. Utemeljene bodo izbire uporabljenih jezikov in tehnik razvoja.

2.1 Uporabniška izkušnja in oblikovanje grafičnega vmesnika

K pojmu uporabniška izkušnja in oblikovanje grafičnega vmesnika sodijo kratice UIX, UI in UX. Kratica UIX (angl. *user interaction and experience*) pomeni uporabniška interakcija in izkušnja. Gre za poimenovanje faze načrtovanja pri oblikovanju uporabniškega vmesnika. Pogosto lahko poleg kratice UIX srečamo tudi kratici UI (angl. *user interface*) in UX (angl. *user experience*). Oblikovalci UI se ukvarjajo z oblikovanjem grafičnega vmesnika. Njihovo delo je predvsem sam izgled vmesnika preko katerega uporabniki komunicirajo z napravo oz. storitvijo. V grobem poznamo konzolne in grafične vmesnike. Slednji bodo v nadaljevanju tudi podrobneje opisani.

Pomembno vlogo imajo tudi UX/UIX oblikovalci. Njihov obseg dela zajema več komunikacije z uporabniki. Pripravljajo poteke interakcij, ankete

na podlagi uporabe vmesnika in raziskujejo kaj pri uporabniški izkušnji deluje in kaj ne. Zavedajo se, da je oblika vmesnika odvisna od ciljne publike in da je potrebno poznavanje človekovega vedenja, druženja, mišljenja, pomnenja ter motivacije. V večjih organizacijah tako strokovnjak za uporabniško izkušnjo razume dele ekonomije, psihologije in sociologije. Z upoštevanjem vseh kriterijev lahko ustvari zelo dobro uporabniško izkušnjo in uporabnike motivira za uporabo.

Pri načrtovanju grafičnega vmesnika za ABML algoritem je motivacija pri uporabi še posebej pomembna, saj gre v osnovi za sistem preko katerega se uporabniki učijo. Enega izmed večjih problemov predstavlja velik nabor podatkov, ki jih je ob uporabi vmesnika potrebno prikazovati.

Za lažje načrtovanje si lahko pomagamo z 10 načeli, ki jih je že leta 1995 zapisal raziskovalec dr. Jakob Nielsen [16]:

1. **Vidljivost statusa sistema**
   Uporabnika je ves čas potrebno obveščati kaj se dogaja s sistemom. Primer so statusne vrstice (brskalniki jih imajo ponavadi čisto spodaj).

2. **Prilagoditev realnemu svetu**
   Sistem naj "govori" uporabniku prijazen jezik. Naj ne uporablja sistemskih izrazov.

3. **Nadzor in svoboda**
   Uporabnik velikokrat izbere akcijo/operacijo pomotoma, omogočeno mora imeti možnost prekinitve in akcijo razveljaviti oz. uveljaviti.

4. **Konsistentnost in standardi**
   Za vsako akcijo je en gumb oz. element. Iste akcije ni možno sprožiti z dvema različnima elementoma.

5. **Izogibanje napakam**
   Vedno preverjamo, če so vnosi pravilni.

6. **Raje prepoznaj kot si zapomni**
   Uporabniku ni potrebno, da si zapomni informacije med različimi prikazi, relevantne informacije se prenašajo samodejno.

7. **Fleksibilnost in učinkovitost**
   Kasneje so ob izvajanju dodane bližnjice, ki pri prvem zagonu niso prikazane.

8. **Estetika in minimalistično načrtovanje**
   Nepomembnih informacij ne prikazujemo. Uporabljamo nevtralne barve, saj tako manj obremenjujemo oči.

9. **Javljanje napak**
   Uporabnik mora biti z napakami seznanjen. Dialog z besedami sporoči napako, le redko uporabimo sistemske kode za sporočanje.

10. **Pomoč in dokumentacija**
    Čeprav se nagibamo, da je sistem enostaven za uporabo, mora biti pomoč in dokumentacija vseeno dosegljiva.

Pri načrtovanju smo se trudili čim bolj držati teh pravil. Dodelali smo razne malenkosti in jih posodobili na novejše načine izvedbe. Namesto statusne vrstice smo uporabili indikatorje nalaganja čez celotno stran. Na ta način se onemogoči, da med procesiranjem pride do napak, ki bi jih lahko sprožil uporabnik s kliki po ekranu. Sistem je zaenkrat preveden v angleški jezik, kasneje pa se mu bo po potrebi dodal tudi slovenski (ali katerikoli drug) jezik. V skladu z 2. načelom se nikjer ne uporabljajo sistemski izrazi. Izgledi so konsistentni, vsaka akcija ima le en prožilni element. V skladu s 5. načelom se pravilnost podatkov preverja že pri uporabniku, tako da do strežnika pridejo samo pravilno vnešeni podatki. Uporabnik je razbremenjen pomnenja, saj se vse potrebne informacije prenašajo samodejno. Gumbi, ki so namenjeni uporabi po prvem zagonu so sprva ustrezno skriti, prav tako so v določenih fazah skrita določena polja. Čeprav je struktura aplikacije zelo

pregledna in razumljiva, je vseeno omogočen ogled pomoči. Pri tem je pomembno, da omogočimo tudi pregled dokumentacije, saj je sistem namenjen za različne podatkovne domene.

2.2 Grafični uporabniški vmesnik pri inteligentnih tutorskih sistemih

Grafični vmesniki (angl. *graphical user interface, GUI*) so nepogrešljivi del pri računalništvu. Začetki takšnega načina vizualizacije segajo že v leto 1973, ko so v podjetju Xerox PARC (Palo Alto Research Center) izdelali prvi vmesnik za namizni računalnik [11]. Kasneje so se le-ti izrazito razvijali in tako smo danes prišli do množične uporabe spletnih vmesnikov. Čeprav se ljudje ne zavedamo, jih venomer uporabljamo in nam dobro služijo. Srečujemo jih povsod, pa naj bo to mobilni telefon, tablica, računalnik, delovni stroj, avtomobil, ura, nadzorne plošče v elektrarni in še bi lahko naštevali. Z njimi lahko upravljamo od enostavnih (npr. klik z miško in odpiranje datoteke) pa vse do zelo zahtevnih operacij (npr. delo s 3D programi in modeliranje), ne da bi se zavedali kaj se v ozadju dogaja s programsko kodo in raznimi aplikacijskimi vmesniki (API).

V praksi se grafični vmesniki delijo predvsem glede na način izvedbe in na vrsto uporabe. Razumljivo je, da bodo pri računalniških igrah ustvarjeni z živahnejšimi barvami in konstruirani z bolj drznimi potezami. Pri vsakdanji uporabi vmesnikov (npr. v pisarnah) pa je nekoliko drugače. Zaradi nenehne uporabe je pomembna uporaba bolj blagih barv, ki ne utrujajo oči. Zelo pomemben je tudi podatek o kontrastu. Svetla podlaga in temno ospredje je po raziskavah najbolj optimalno za naše oči [1]. Poleg vseh podrobnih lastnosti barv in izgleda je pomembna tudi smiselnost postavitve elementov. Kako uspešni smo bili pri izdelavi, se lahko enostavno prepričamo že s prvim pogledom na vmesnik. Dober vmesnik nam da hitro vedeti kaj nam ponuja in kaj zahteva od nas. Logično in jasno sporoča potek komunikacije, informacije so lahko berljive in so predstavljene razumljivo ter razločno. Malo večji izziv

nastane pri prikazu večjega števila podatkov. Takrat je potrebno biti zelo previden, da ne prenasičimo prikaza in hkrati ohranimo namen in sporočilo. Poleg tega, da mora vse skupaj izgledati enostavno in lepo, je pomembna tudi dobra interakcija, vse skupaj pa nam prinese dobro uporabniško izkušnjo.

Pot do dobrega vmesnika si lahko olajšamo z uporabo nekaterih pomembnih načel [6]. Bistvena bodo na kratko našteta in opisana.

Pri načrtovanju moramo upoštevati:

1. **Barve**
   Poznamo več barvnih sestavov. Med bolj znanimi so: analogni sestav, komplementarni sestav in barvne triade. Za uporabo v grafičnem vmesniku je bil izbran analogni sestav (uporaba sosednjih barv) (Slika 2.1).

2. **Oblike**
   Ker gre za prikaz podatkov, so bile izbrane osnovne oblike pravokotnikov, saj bodo informacije tako bolj pregledne. Pri izboru za izražanje vrednosti, večji oz. manjši, pa so primerni trikotniki, saj s svojim vrhom enostavno nakazujejo izbor.

3. **Poravnave**
   Elementi so ravnani horizontalno in vertikalno, saj s tem pridobimo boljši pregled nad vrsticami ter atributi tabele.

4. **Enostavna zasnova**
   Manj je več. Osredotočati se je potrebno na glavne lastnosti izdelka. Preveč nepotrebnih dodatkov lahko uporabnika odmakne od bistva.

5. **Ponavljane vzorcev**
   Z uporabo istih barv in pozicij kontrolnih elementov kot so gumbi in indikatorji, uporabniku olajšamo interakcijo. Vmesnik pa bo bolj pregleden in enoten.

6. **Velikosti elementov in objektov**
   Večji kot je element, večjo pozornost mu bomo namenili. Glavni po-

datki morajo biti v središču pozornosti, dodatne informacije so zapisane v manjšem formatu.

7. **Smiselnost vsebinske razporeditve elementov**
   Grupiranje elementov v mini "plošče" vsebinsko razdeli celoten prostor, hkrati pa poveča preglednost celotne strukture.

Slika 2.1: Izbran analogni sestav v barvnem krogu.

Izdelava vmesnika se je začela z vprašanjem, katere vrste vmesnik bi bil primeren glede izbrane tematike. Ker se trenutno čedalje več aktivnosti odvija v tako imenovanih oblakih, je bila ideja o spletnem vmesniku nekako najbolj primerna. Poleg tega, da bo upravljanje namesto v konzoli zdaj potekalo preko gumbov in grafičnih elementov, pa pridobimo še oddaljeni dostop in večuporabniško spletno aplikacijo. Ta je v osnovi realizirana z uporabo označevalnega jezika HTML in CSS. Dodana pa sta še jezika za dinamičnost strani, JavaScript in AJAX, ki kontrolirata vse akcije na strani.

AJAX skripta (Slika 2.2) s spletnimi zahtevami preko API vmesnika pridobiva strežniške podatke, ki jih nato procesira v JavaScript in na koncu v HTML ter CSS. Vsemu skupaj je dodano še nekaj knjižnic za oblikovanje (CSS), ter JavaScript knjižnica za komunikacijo s strežnikom in uporabo drugih funkcij. Pri izdelavi vmesnika so bila uporabljena različna odprtokodna orodja za urejanje:

- **Komodo Edit**
  Urejanje spletnih jezikov HTML, CSS, JavaScript, AJAX.
- **GIMP**
  Urejanje slik, ikon in drugih grafičnih elementov.
- **Google Chrome**
  Testiranje vmesnika.
- **Mozilla Firefox**
  Testiranje vmesnika.

```
function getCritExamplesABML() {
    $("#progressStatus").html("Pripravljam podatke ...");
    var formData = {
        'sessionID': sessionID
    };
    $.ajax({
        type: 'POST',
        url: 'api/getCriticalExamplesABML.php',
        data: formData,
        dataType: 'json',
        encode: true
    }).then(handleSuccess, handleError);
}
```

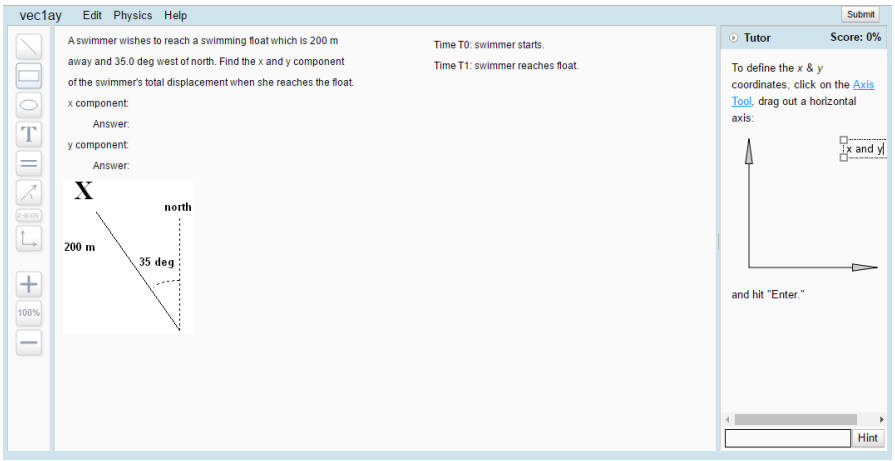
Slika 2.2: Primer AJAX spletne zahteve za pridobivanje podatkov.

2.2.1 Analiza obstoječih grafičnih vmesnikov inteligentnih tutorskih sistemov

Kot nadomestek učiteljev za poučevanje ena na ena se je v praksi začela uporabljati alternativa: računalniški sistemi, katerih zgradba stremi k čimbolj podobnim lastnostim kot so učiteljeve. Učencu poleg razlage nudi tudi reševanje primerov, ki jih po potrebi pojasni.

Poznamo več vrst računalniških sistemov za poučevanje. Najprej se je razvila uporaba tako imenovanih CAI sistemov. V praksi se izkaže, da je njihova uporaba bolj uspešna kot učenje v razredih, ter še vedno manj uspešna kot učenje učitelja ena na ena. Njihova glavna pomanjkljivost je, da se niso zmožni prilagoditi specifičnim potrebam posameznika. Primer takšnih sistemov so izobraževalne aplikacije za tablične računalnike. Teh je na tržišču ogromno.

Bolj napredni sistemi za poučevanje so ITS. Sposobni so graditi model učenca, kar potem omogoča prilagajanje vsakemu učencu posebej. Kot prvi primer takšnega sistema navedimo Andes Physics Tutor[1] (Slika 2.3), ki se uporablja za učenje fizike. Izvajanje sistema poteka v celoti preko brskalnika.

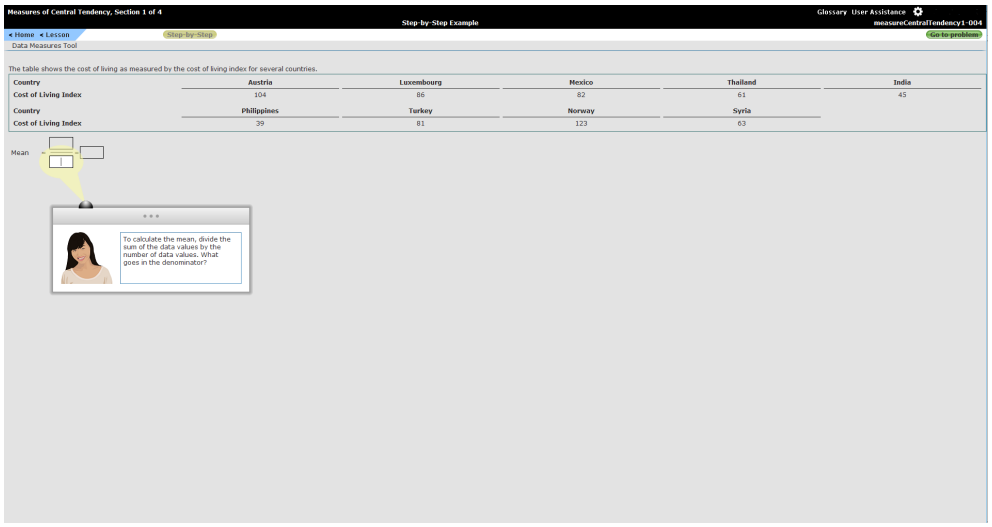


Slika 2.3: Sistem za poučevanje Andes Physics Tutor.

[1] http://www.andestutor.org/

Trenutno najbolj dovršen in priljubljen ITS je Carnegie Learning,[2] ki se za poučevanje matematike množično uporablja v ZDA. Če ga primerjamo z Andes Physics Tutor, ugotovimo, da je veliko bolj dodelan, saj ga (tudi s pomočjo znanstvenih raziskav) izpopolnjujejo že 20 let. V ZDA ga v praksi uporablja okrog 650.000 osnovnošolcev in srednješolcev ter okrog 150.000 študentov. Z več obsežnejšimi raziskavami so pokazali, da so učenci, ki so se učili s pomočjo kognitivnega tutorja Algebra 1, ki sicer predstavlja le del celotnega sistema, v povprečju dosegali za 85 % boljše rezultate.

Sam potek učenja poteka v posebnem oknu, ki nam ga odpre brskalnik (Slika 2.4). Za delovanje programa potrebujemo nameščeno različico Jave.[3] Za primerjavo: pri Andesu, za delovanje potrebujemo le brskalnik, brez dodatnih namestitev.



Slika 2.4: Sistem za poučevanje Carnegie Learning.

Preizkusili smo oba sistema ter ju vzeli v primerjavo. Načeloma primerjava dveh sistemov z različno učno domeno ni enostavna, a nam kljub temu lahko pomaga, da lažje ugotovimo vsaj nekaj osnovnih lastnosti obeh sistemov. Pri obeh opazimo nekaj bistvenih razlik. Že pred zagonom je opazno,

[2] `https://www.carnegielearning.com/`

[3] `https://java.com/`

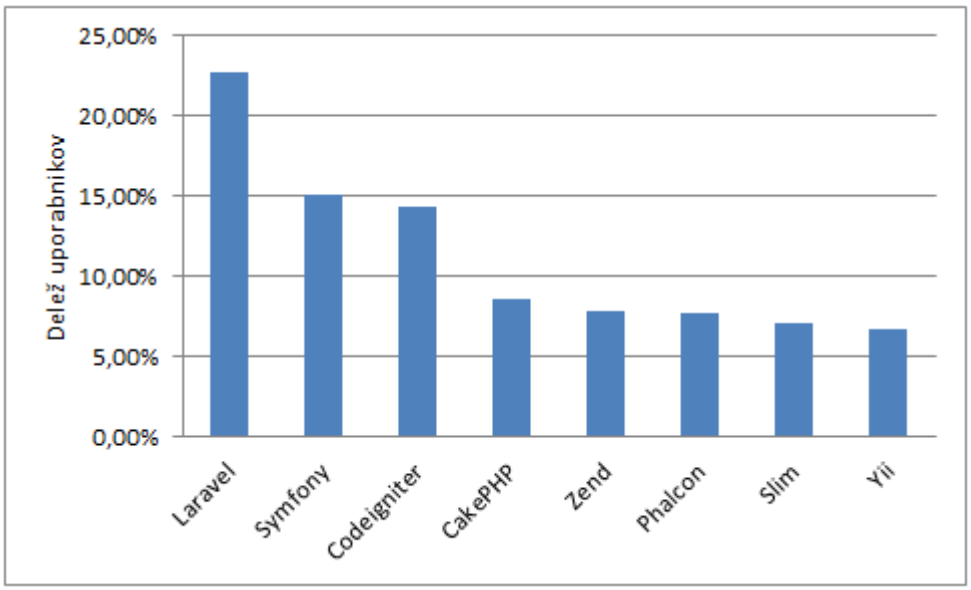
da pri Andes sistemu, ne potrebujemo svojega uporabniškega računa, temveč je dovolj, če vpišemo samo svoje ime. Pri Carnegie Learning je uporabniški račun neobhoden. Na podlagi tega se potem gradijo tudi specifični modeli za vsakega učenca posebej. Sledi princip pred začetkom učenja, ki je pri obeh približno podoben. Pri obeh si izberemo tematiko iz katere bi se radi učili, nato pa se nam prikaže dejanska naloga. Andes Physics Tutor se tu prikaže v malo slabši luči. Grafični vmesnik je sicer kar pregleden, vendar malce starejšega izgleda. Poleg tega na strani pogrešamo bolj enostavno sprotno pomoč. V primerjavi s pomočjo pri Carnegie Learning je dosti manj pregledna in razumljiva. Sicer opazimo, da v podstraneh vsebuje tudi video pomoč, a je uporaba vseeno veliko bolj kompliciranа.

Opazno je, da se Carnegie Learning uporablja v večji meri kot Andes Physics Tutor, saj se zaradi večjega števila uporabe tudi nenehno razvija ter izboljšuje. Posledično to pomeni tudi boljšo izvedbo sistema in kakovostnejšo interakcijo. Še zlasti je pomembna opazka, da nam oba sistema omogočata prikaz namigov, le da imamo pri sistemu Carnegie Learning tudi možnost prikaza rešitve korak za korakom.

2.2.2 Tehnologija grafičnega vmesnika

Pri odločanju katero tehnologijo bomo uporabili za izdelavo grafičnega vmesnika si najprej postavimo mejnike, do kakšne mere se bo vmesnik v prihodnje razvijal. Vprašamo se kako je z nadgrajevanjem in celotno strukturo, ali se bodo pogosto dogajale spremembe podatkovnih izpisov, menjav celotnih tem prikazovanja ali pa bo šlo za bolj ali manj fiksno strukturo, ki bo potrebovala manjše spremembe, konkretnih posodobitev pa ne bo v načrtu. Razlika se opazi tudi pri velikosti strukture. Če bi želeli aplikacijo narediti lažjo za posodobitve, bomo potrebovali sistem za enostavno nadgradnjo celotnega grafičnega vmesnika. Tukaj se stvari občutno razvejajo, saj to pomeni, da vstopi še kup drugih funkcij in dejavnikov, ki na to vplivajo. Potrebno je zgraditi celoten sistem nadgradenj. V veliki meri nam pridejo v pomoč že narejene strukture, tako imenovana ogrodja (angl. *frameworks*), ki olajšajo

delo, vendar zahtevajo poznavanje natančnega delovanja celotnega sistema. Potrebno je vedeti tudi, da je raba takšnega ogrodja povezana s programskim jezikom, ki teče na strežniku. Takšni sistemi tako obstajajo za strežniške jezike PHP, Python, Ruby in podobne. Primer nekaj PHP ogrodij [13] glede na priljubljenost je prikazanih v grafikonu (Slika 2.5, vir: [19]), podrobnejši opis sestave ogrodij pa se nahaja v podpoglavju 2.4.1.



Slika 2.5: Nekaj PHP ogrodij glede na priljubljenost v letu 2015.

V našem primeru ni potrebe po uporabi ogrodja in kompleksni strukturi vmesnika, saj se pri nadaljnjih nadgradnjah pričakujejo le manjše spremembe, portal pa bo izključno namenjen samo tematiki argumentiranega strojnega učenja z uporabo grafičnega vmesnika. Zaradi možnosti uporabe več različnih domen učnih podatkov in s tem tudi različnega števila argumentov, ki se uporabljajo pri prikazu, je sistem narejen tako, da se grafični vmesnik samodejno prilagaja glede na različne podatke. To pa nam tudi poenostavi delo, saj ob menjavi domene oz. učnih podatkov ni potrebnega nobenega vzdrževanja ali posodobitve sistema.

2.3 Argumentirano strojno učenje

Argumentiratno strojno učenje oz. ABML omogoča pridobivanje znanja na podlagi izkušenj s pomočjo argumentov in temelji na iskanju pravil v učnih podatkih. Pomembna razlika med običajnim in argumentiranim strojnim učenjem [15] je v določanju oz. iskanju pravil. Običajno strojno učenje nad danimi podatki sklepa pravilo, ki pa ga v resnici "ne razume". Pri uporabi argumentiranega strojnega učenja pa imamo možnost vplivati na računalniško iskanje pravil s pomočjo argumentov domenskega strokovnjaka, ki jih v interaktivnem postopku lahko vpelje v domeno. Tako lahko pripomoremo k oblikovanju bolj logičnih pravil in računalniku "obrazložimo" zakaj je temu tako. Računalnik pa je sposoben samodejno poiskati ustrezne protiprimere, ki domenskemu strokovnjaku omogočijo izboljšanje podanih argumentov.

Tako pridobljena pravila so praviloma točnejša in bolj skladna z ekspertnim znanjem [14, 8, 9]. Zaradi navedenih lastnosti obstaja možnost, da bi argumentirano strojno učenje lahko uspešno uporabili pri gradnji inteligentnih tutorskih sistemov [27]. Seveda brez domenskih ekspertov pri gradnji tovrstnih sistemov ne gre, zato na začetku potrebujemo tudi strokovnjaka, ki lahko čim bolj natančno določi vrednosti razreda primerov v učnih podatkih, kot so denimo bonitetne ocene podjetij, razlaga matematičnih primerov in podobno. Hkrati pa lahko domenski strokovnjak v sistem vpelje dodatne koncepte, ki so nato lahko predmet poučevanja [17].

2.3.1 Interaktivna zanka za zajemanje znanja

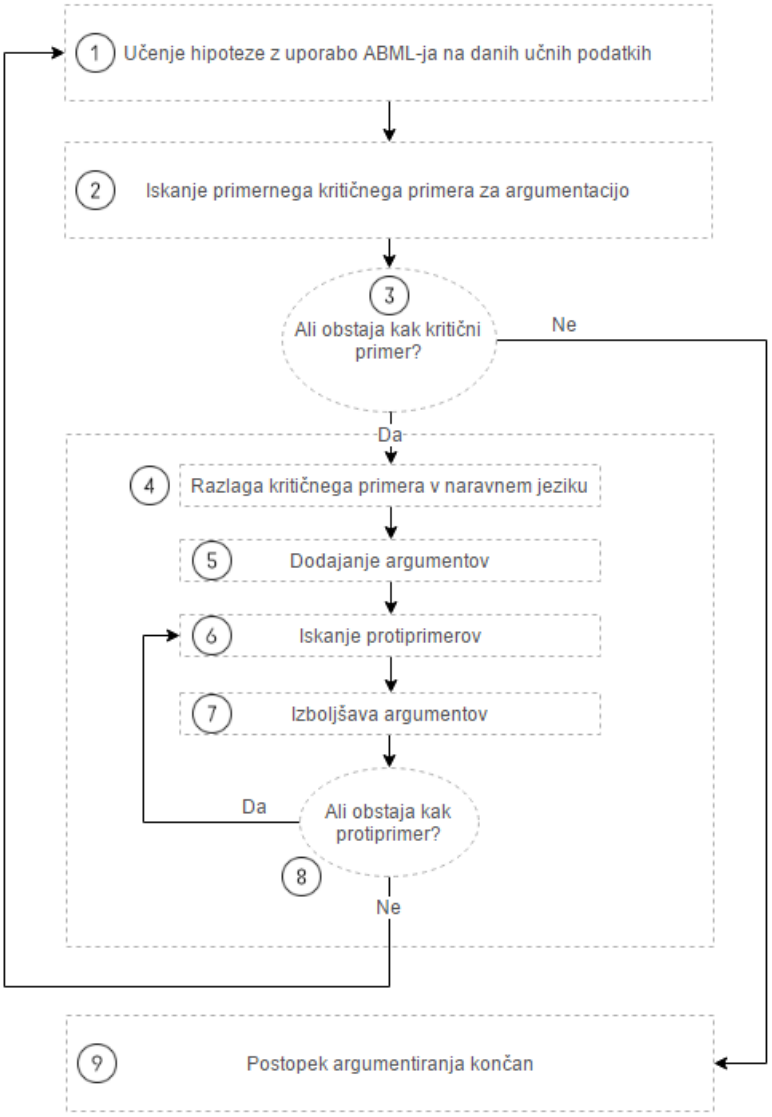
Interaktivna zanka za zajemanje znanja (v nadaljevanju: interaktivna zanka) je eden izmed dveh ključnih delov ABML algoritma. V ABML je vključena v kombinaciji s spremenjenim algoritmom strojnega učenja za indukcijo pravil CN2 [15], ki za delovanje lahko uporablja tudi podane argumente. Zajemanje znanja poteka iterativno in v interakcijo vključuje znanje domenskega strokovnjaka [9, 14], v kontekstu inteligentnih tutorskih sistemov pa lahko interakcija poteka tudi z učencem [25]. Uporabnik (npr. domenski ekspert) s

pomočjo interaktivne zanke pridobi kritične primere in jim poda argumente za izboljšavo pravil, kot je opisano v nadaljevanju in ilustrirano na sliki 2.6.

- V prvem koraku ABML algoritem nad danimi učnimi podatki postavi hipotezo. Naučeni model oz. hipoteza je predstavljena kot množica pravil, ki nastanejo kot rezultat delovanja algoritma ABCN2.
- V drugem koraku se na podlagi pridobljenih pravil poišče kritične primere. Gre za učne primere, ki jih trenutna pravila narobe uvrstijo oz. klasificirajo. Algoritem nam lahko poda več kritičnih primerov, razvrščeni so po t. i. stopnji kritičnosti.
- Algoritem predstavi kritični primer ekspertu. Če kritičnih primerov ni, zaključimo s postopkom argumentiranja (korak 3).
- Ekspert primer pregleda in ga razloži v naravnem jeziku (korak 4). V primeru, da ne najde ustrezne razlage, lahko zahteva nov kritični primer. S pomočjo uporabniškega vmesnika, se ekspertovo razlago kritičnega primera vnese v sistem (korak 5).
- Izvede se iskanje protiprimerov (korak 6), ki nakazujejo, ali je razlaga kritičnega primera dovolj dobra ali pa jo je potrebno še izboljšati. V slednjem primeru lahko ekspert svojo razlago dopolni oz. spremeni (7. korak).
- Algoritem upošteva spremembe argumenta in ponovno izvede iskanje protiprimerov (korak 6). Če ni najdenih protiprimerov, algoritem poišče nov kritični primer.
- Ponovno se izvede učenje pravil (1. korak), pri čemer se upoštevajo ekspertni argumenti. Le-ti se vselej nanašajo na kritične primere.

Postopek argumentiranja se zaključi (korak 9), ko ni več kritičnih primerov. Načeloma lahko pri koraku 6 še vedno obstajajo protiprimeri, vendar ekspert ne želi več spreminjati argumentov, ker bi s tem dobil zelo dolge argumente,

ki ne bi bili več dovolj splošni. V tem primeru lahko zahteva nov kritični primer. Meja stopnje kritičnosti, ki jo omenimo pri koraku št. 2, je arbitrarno določena. Lahko jo prilagajamo in s tem določamo mejo kritičnosti.



Slika 2.6: Postopek zajemanja znanja s pomočjo interaktivne zanke.

Postopek zajemanja znanja s pomočjo interaktivne zanke je dodatno ilustriran s primerom v nadaljevanju [26].

oseba	**temperatura**	**cepljen**	**kašljanje**	**glavobol**	**gripa**
1	da, visoka	da	da	ne	ne
2	da, zelo visoka	ne	ne	da	da
3	da, visoka	ne	da	ne	da
4	ne	da	ne	ne	ne
...	...	...	...	...	...

Tabela 2.1: Učni podatki.

Nad danimi podatki (Tabela 2.1) bi računalnik ustvaril model s pravilom: *Če oseba nima zelo visoke vročine, potem nima gripe.* To pa nikakor ni skladno z vsemi primeri. Oseba 3 nima zelo visoke vročine, a iz podatkov vidimo, da ima gripo. Pojavi se vprašanje, zakaj pride do tega. Metoda strojnega učenja (angl. *machine learning method*, v nadaljevanju: ML) v tem trenutku ta primer prikaže kot kritični primer, katerega mora nato domenski ekspert argumentirati. Ta mu kasneje kot argument lahko navede, da ima oseba gripo zaradi temperature, ki je višja od normalne: *Oseba 3 ima gripo, ker ima vročino.* ML nato v uporabi z argumentom, izpelje novo pravilo: *Če ima oseba temperaturo, potem ima tudi gripo.* Ponovno se pojavi neskladje dobljenega pravila in učnih podatkov. Namreč, glede na pravilo, v učnih podatkih najdemo protiprimer št. 1, kjer ima oseba vročino, nima pa gripe. Ekspert ima nato omogočeno primerjavo primera 1 s primerom osebe 3. S primerjanjem poskuša najti razlog, zakaj oseba 1 nima gripe. Odgovor najde pri podatku o cepljenju. Oseba 1 je bila proti gripi cepljena, medtem ko oseba 3 tega ni bila deležna. ML ponovno sproži postopek sklepanja novega pravila, uporabi podane argumente in določi novo pravilo: *Če ima oseba temperaturo in ni bila cepljena proti gripi, potem ima gripo.* Z uporabo tega pravila pa nato potrdi tudi primer osebe 2.

Iz opisa primera lahko opazimo, da je ena izmed ključnih prednosti interaktivne zanke ravno avtomatsko iskanje protiprimerov, ki ekspertu dodatno pomagajo pri razlagi kritičnega primera oz. pri podajanju argumentov. Kot

dodatno možnost ima ekspert omogočeno tudi dodajanje novih konceptov oz. atributov, s katerimi si olajša delo. Z vpeljavo atributa "Simptomi gripe", lahko kasneje enostavneje argumentira protiprimere. Takšna vpeljava bi potem pomenila tudi drugačno sklepano pravilo: *Če ima oseba temperaturo in ima simptome gripe, potem ima gripo.*

Pri tem pa vedno obstaja možnost, da se v danih podatkih pojavi napaka. Domenski strokovnjak lahko s pomočjo samodejno najdenih kritičnih primerov in protiprimerov to tudi opazi in takšne primere popravi.

2.4 Komunikacija vizualne strani s strežnikom

Dandanes je v uporabi več različnih načinov komunikacij. Kateri način komunikacije bomo izbrali, je odvisno od tega kaj želimo in kakšen bo naš produkt. Pri izbiri komunikacije govorimo o izbiri programskega jezika in tehnike sporočanja podatkov. Vsak jezik ima svoje prednosti in slabosti. Pomembno je, da že pred začetkom programiranja vemo, kaj je naša tematika in kaj bomo ponujali uporabnikom. Pomembno vlogo ima tudi strežnik na katerem se bodo izvajali storitveni servisi, kot so razni strežniški procesi in spletni strežnik za serviranje spletne vsebine. Velikokrat smo namreč glede izbire omejeni s strani ponudnika gostovanja, izjema je, če smo lastniki strežnika sami. To bi potem pomenilo, da smo pri izbiri načina komunikacije povsem svobodni in je potrebno, da za vse skrbimo sami ali pa imamo vzdrževalca, ki namesto nas poskrbi tako za varnost kot tudi za pravilno delovanje strežnika in morebitne posodobitve. V nadaljnji razlagi je vizualna stran v kontekstu uporabljena kot spletni odjemalec (brskalnik), strežnik pa kot "ABML servis", ki se v tem primeru poganja na istem strežniku kot spletni strežnik za serviranje spletne vsebine.

Načini komunikacije se razlikujejo glede na namen uporabe aplikacije. Aplikacije, namenjene predvsem prikazu in shranjevanju manjšega števila podatkov, navadno niso procesorsko zelo zahtevne. Pri takšnih izvedbah je zadovoljiva že majhna procesorska moč, saj se bo poleg procesiranja prikaza

informacij, izvajal le še proces podatkovne baze (DB, angl. *database*) in vmesnika za dostop do podatkovnih baz (MySQL, angl. *management system based on structured query language*). Pri teh aplikacijah lahko izberemo jezik, ki ne nudi dovolj dobre opore pri računanju večjih računskih problemov. Povsem nasproten primer so aplikacije, ki v svojem delovanju uporabljajo kompleksnejše računske operacije in so posledično tudi bolj zahtevne. Takšnim sistemom je potrebno zagotoviti boljši strežnik, ki bo podpiral uporabo primernih jezikov, s tem pa tudi povečamo hitrost in zmanjšamo čas čakanja ob procesiranju.

Uporabljeni ABML algoritem, integriran v strežniški servis, sodi med računsko kompleksnejše sisteme. Vsebuje zahtevne operacije strojnega učenja, ki se nato uporabljajo v kombinaciji z lokalnim podatkovnim sistemom, ki za vsakega uporabnika hrani njegove podatke. ABML je implementiran v odprtokodnem programskem paketu Orange oz. Orange Data Mining [5] in nam omogoča izvedbo podatkovne analize. Analiza se nato uporabi pri procesiranju, potek interakcije pa se zapiše v datoteke. Samo učenje pravil je nad večjimi množicami podatkov lahko časovno zelo potratno in za delovanje potrebuje kar nekaj procesorske moči.

Seveda je pri preoblikovanju kode pomembna tudi čim učinkovitejša optimizacija, saj s tem pridobimo na krajšem času izvedbe in manjši porabi pomnilnika. ABML algoritem je napisan v Python [20] programski kodi. Prvotno delovanje je realizirano z neskončno zanko, v kateri se izvajajo podzanke za vnos in izpis ključnih podatkov sistema, ki služijo za komunikacijo z uporabo konzole. Pri preoblikovanju je potrebno takšno zanko odstraniti, v nasprotnem primeru bi to pomenilo, da bi za n uporabnikov, imeli n tekočih procesov ABML programa. To vsekakor ni dobra rešitev. Optimalna modifikacija je izvedljiva v smeri delitve procesa na manjše niti. Tako lahko odpravimo zanko in ob vsaki novi povezavi ustvarimo novo nit za uporabnika. Nit je aktivna tako dolgo, dokler se v celoti ne izvede podfunkcija algoritma. To lahko storimo zaradi preoblikovanja funkcij v podfunkcije, ki se izvedejo hitreje in omogočajo programsko delitev primerno za uporabo z

vizualno stranjo preko spletnega strežnika, s procesiranjem v realnem času.

Zaradi uporabe grafičnega vmesnika direktno z algoritmom v realnem času je pomembno, da komunikacija poteka hitro in kakovostno [3]. Napake pri prenosih niso sprejemljive, saj nam le še dodatno povečujejo čas izvajanja, kar pa z vidika uporabnikov ni najbolj sprejemljivo. Kakovost se odraža tudi s sporočanjem dejanskega stanja komuniciranja, tako da je uporabnik vseskozi seznanjen o poteku izvajanja. Z upoštevanjem vseh teh napotkov, lahko kot rezultat dobimo dobro vizualizirano različico ABML-ja [23]. Vizualizacija se nanaša na grafični oz. slikovni multimedijski prikaz podatkov, s katerim ustvarimo kvalitetno strukturirano vizualno podobo. Ključen rezultat tega je v izboljšanju miselnih procesov in s tem tudi poudarjanju relevantnih podatkov, kar nam prinese kvalitetnejše informacije [10].

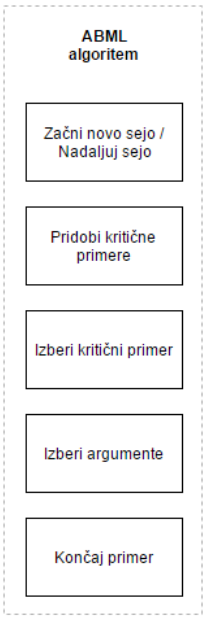
V naslednjem podpoglavju PHP in Python bo tudi razloženo zakaj je to boljše in kako dejansko delujejo medsebojne povezave modulov v celotem sistemu.

2.4.1 PHP in Python

Pri izbiri strežniškega dela jezika je bilo potrebno nujno uporabiti tudi Python programski jezik, saj je z njim realiziran celoten algoritem (v nadaljevanju: interaktivni učni algoritem), ki ga je Matevž Pavlič nadgradil v svojem magistrskem delu [17]. Python spada med enostavnejše programske jezike. Namenjen je začetnikom kot tudi zahtevnim uporabnikom in je vsekakor primeren za uporabo v kompleksnih sistemih kot je strojno učenje [18]. S svojimi podatkovnimi tipi upravlja popolnoma dinamično in podpira funkcionalno, proceduralno, strukturirano in objektno orientirano programiranje [21].

Za delovanje spletnega vmesnika je potrebno ustvariti nekakšen sistem, ki bi programsko kodo interaktivnega učnega algoritma preoblikoval, da bo možna sprejemati in pošiljati podatke [12]. Hkrati pa je zaželeno, da bi interaktivni učni algoritem deloval za več uporabnikov sočasno. Rešitev bi dosegli z integracijo učnega algoritma v Python spletni servis, ki bi kontroliral komunikacijo med odjemalcem ter algoritmom. Za komunikacijo servisa in

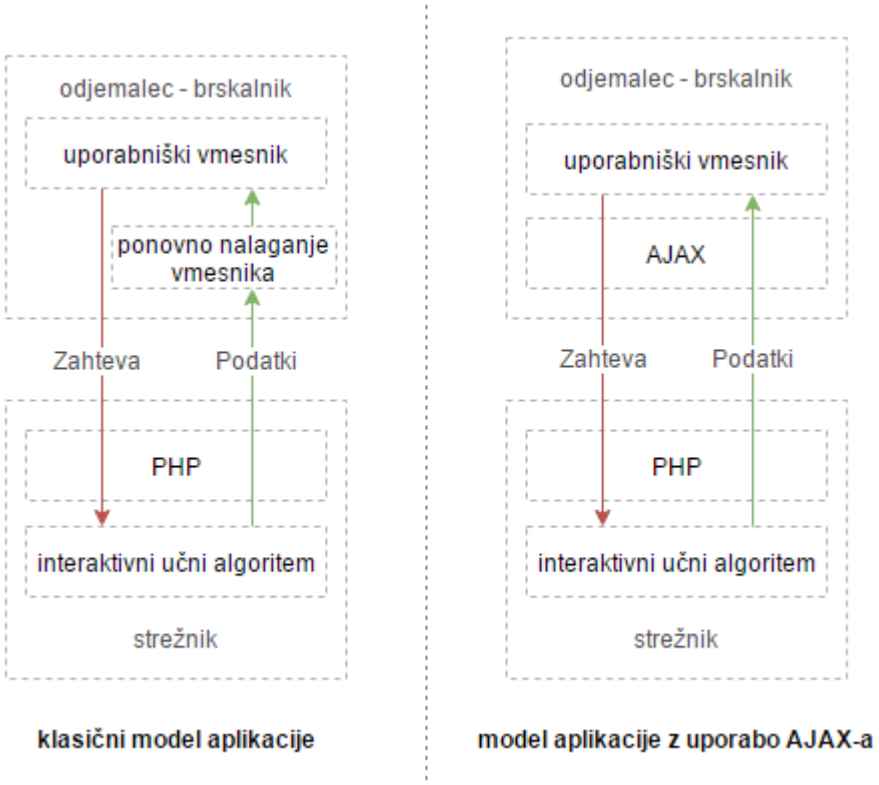
odjemalca je potrebno izdelati tudi lasten protokol, ki skrbi za obliko prenosa podatkov. Ta nam omogoča komunikacijo na takšni ravni kot jo želimo sami. Pri tem imamo zelo odprte možnosti, saj je vse funkcionalnosti potrebno posebej določiti. Ob določanju funkcij s katerimi se kličejo akcije, smo si lahko pomagali z dekompozicijo interaktivnega učnega algoritma, ki je bil razdeljen na manjše module (Slika 2.7). Protokol ima definirane ukaze, vsak ukaz pa se uporablja za kontroliranje manjšega modula oz. funkcije v omenjenem algoritmu.



Slika 2.7: Dekompozicija interaktivnega učnega algoritma na manjše module.

Del komunikacije so tudi AJAX zahteve [7], ki se pri spletnem programiranju definirajo kot kontrolne funkcije, ki dinamično oz. asinhrono upravljajo z zahtevami po podatkih. Ta lastnost nam omogoča dinamičnost prikazovanja podatkov, brez da bi se pošiljala zahteva po ponovnem nalaganju celotne spletne strani. Struktura modela aplikacije brez in z uporabo AJAX funkcije (Slika 2.8) je zelo podobna. V shemi se modela razlikujeta pri funkciji pošiljanja. Brez uporabe AJAX-a se v brskalniku ob prejetju novih podatkov

ponovno naloži celoten uporabniški vmesnik. To seveda zahteva svoj čas, saj se vse slike in elementi ponovno prenašajo iz strežnika. V nasprotnem primeru se vnovičnemu prenašanju izognemo, saj se po prejetju podatkov osveži le spremenjena vsebina.



Slika 2.8: Primerjava aplikacije brez in z uporabo AJAX-a.

Poleg integracije algoritma v servis, je bilo potrebnih še nekaj sprememb pri že obstoječih metodah ABML učne zanke. Kljub vsem spremembam nam je uspelo vseeno ohraniti vse funkcionalnosti, tudi tiste, ki obsegajo celotne zapise zgodovine o uporabi vsakega uporabnika. Iz njih je kasneje možno določiti profil uporabnika in oceniti njegov potek učenja.

Strežniški servis je v Python jeziku realiziran s pomočjo komunikacijske vtičnice oz. vmesnika (angl. *socket*). Pri tem se vtičnice uporabljajo kot končne točke dvosmernega komunikacijskega kanala in veljajo kot uni-

katen identifikator [22]. Vtičnice lahko komunicirajo med različnimi procesi v lokalnem računalniku ali pa med različnimi procesi različnih lokacij čisto na drugem koncu planeta. V Python skripti vtičnici definiramo domeno v kateri bo delovala in katera vrata (angl. *port*) bo uporabljala za komunikacijo. Nastavljena domena "localhost" pomeni delovanje vtičnice na lokalnem računalniku, številka vrat pa sporoča, da se bo zajemal le tisti omrežni promet, ki bo prispel na ta vrata.

Za prenos podatkov lahko vtičnici nastavimo TCP (angl. *transmission control protocol*) ali UDP (angl. *user datagram protocol*) prenosni protokol. Podatki ABML sistema so za delovanje grafičnega vmesnika zelo pomembni, zato za prenosne poti nastavimo prenosni protokol TCP, saj nam omogoča zanesljiv prenos.

TCP	UDP
Zanesljiv	Nezanesljiv
Pravilen vrstni red podatkov	Vrstni red podatkov odvisen od povezave, ni nujno da bo pravilen
Robustna sestava	Enostavna sestava
Potrjevanje paketkov	Potrjevanja ni
Počasnejši prenos	Hiter prenos
Večja podpora pri varnosti	V splošnem pri varnosti slabši
...	...

Slika 2.9: Kratka primerjava TCP in UDP protokola za prenos podatkov.

V primerjavi (Slika 2.9) z UDP protokolom se podatki ob neuspešnem pošiljanju pošljejo ponovno, ob uspešnem prejetju pa se sporoči status o sprejetem paketu. Poleg tega je pri TCP prenosnem protokolu omogočen tudi pravilen vrstni red podatkov, ki v povezavi s potrjevanjem paketkov, zagotavlja prenos vseh paketkov brez napak. Čeprav je robustnejši in malo počasnejši je za takšne namene rabe pošiljanja veliko bolj primeren kot UDP. Slednji se uporablja bolj za prenos manjših datotek, kjer ni potrebna garancija o dostavi paketka.

V nadaljevanju bodo opisani še ključni elementi komunikacije med programskima jezikoma PHP in Python. Pomembno je zagotavljati, da podatki pravilno prehajajo po protokolu. Za format, v katerem se podatki prenašajo, je bilo najenostavneje izbrati kar JSON notacijo, saj je struktura podatkov zelo jasna in enostavna. Notacija je sestavljena iz parov v kombinaciji ključ: vrednost (Slika 2.10a). Podpira uporabo tako številskih kot tudi besedilnih vrednosti. Omogočena pa nam je tudi uporaba večnivojskega gnezdenja podatkov (Slika 2.10b).

```
{
    "Ključ 1": 1,
    "Ključ 2": 2,
    "Ključ 3": 3
}
```

(a) Osnovna notacija.

```
{
    "id_podjetja1": {
        "ime": "Podjetje 1",
        "kraj": "Ljubljana",
        "dejavnost": "Računovodske dejavnosti"
    },
    "id_podjetja2": {
        "ime": "Podjetje 2",
        "naslov": "Krško",
        "dejavnost": "Informacijske dejavnosti"
    }
}
```

(b) Gnezdeni podatki.

Slika 2.10: Prikaz podatkov v JSON notaciji.

Python je kot servis potrebno na strežniku zagnati. Ustvari se proces, ki spremlja mrežni promet na TCP vratih 8888. Številka vrat je arbitrarna in jo je pred zagonom možno tudi spreminjati. Ob zagonu se v konzoli sistema (Slika 2.11) najprej izpiše status s številko vrat, preko katerih bo potekala komunikacija med spletnim strežnikom in interaktivnim učnim algoritmom. Pri vsaki vzpostavitvi nove povezave v konzoli vidimo časovni žig, naslov povezanega odjemalca in akcijo, ki se bo izvajala. Načeloma teh zapisov kasneje ne bomo potrebovali, so pa bili v pomoč pri razvijanju sistema. Python skripte se lahko nahajajo tudi na čisto drugem strežniku kot skripte za spletno aplikacijo, saj delujejo neodvisno ena od druge. To je tudi prednost, saj imamo lahko za več grafičnih vmesnikov zagnan samo en servis, do katerega pa lahko dostopamo iz različnih lokacij spletnih strežnikov. S takšnim načinom izvedbe je v povezavi s spletnim strežnikom možno izdelati tudi mobilno ali katerokoli drugo aplikacijo, saj je v sistemu popolnoma razvit sistem za zahteve API. Pomembno prednost imamo tudi pri posodobitvah. Ker gre za sistem, ki je lahko dostopen preko spleta, nimamo težav s posodobitvami, saj se celotna struktura shranjuje na strežniku. Odjemalec je le aplikacija, ki nam je v pomoč pri odpiranju spletnih strani in aplikacij. Za delovanje spletnega strežnika uporabimo odprtokodno orodje Apache,[4] ki podpira izvajanje PHP jezika.

Z ustreznimi ukazi, poslanimi preko PHP skript (Slika 2.12), kontroliramo stanja ABML servisa. Velja omeniti tudi večnitno funkcionalnost servisa, kar nam omogoča neodvisno delovanje algoritma za več uporabnikov hkrati.

[4] `https://httpd.apache.org/`

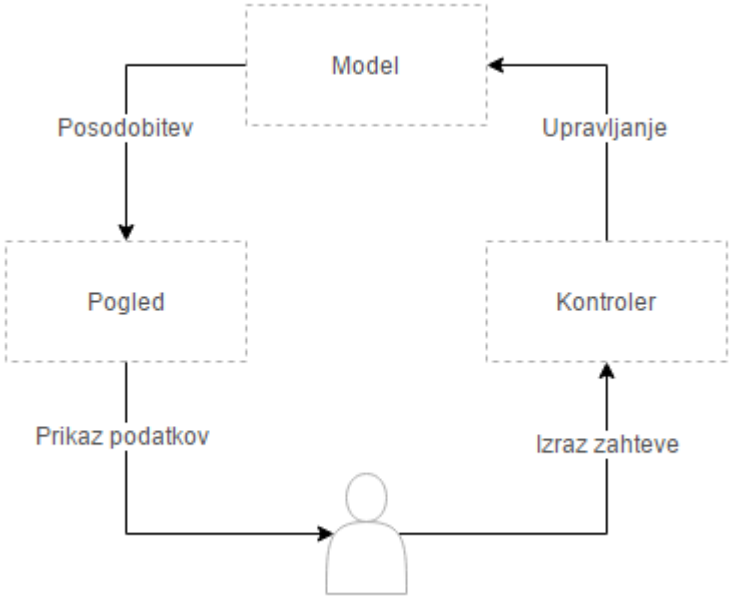
```
starting up on localhost port 8888
----------------------------------------------------------------------------------------------------
06.09.2016 - 12:52.13 || Connected with 127.0.0.1:56868
06.09.2016 - 12:52.13 || 127.0.0.1:56868 () -> Received "ID/-/eb72cd4a"
06.09.2016 - 12:52.13 || 127.0.0.1:56868 (eb72cd4a) -> Received "start_ABML/-/"
**********************************************************************
Learning rules...
06.09.2016 - 12:52.15 || No more data from 127.0.0.1:56868 (eb72cd4a)
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
06.09.2016 - 12:52.15 || Connected with 127.0.0.1:56869
06.09.2016 - 12:52.15 || 127.0.0.1:56869 () -> Received "ID/-/eb72cd4a"
06.09.2016 - 12:52.15 || 127.0.0.1:56869 (eb72cd4a) -> Received "get_criticals_ABML/-/"
**********************************************************************
Finding critical examples...
06.09.2016 - 12:52.19 || No more data from 127.0.0.1:56869 (eb72cd4a)
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
06.09.2016 - 12:52.59 || Connected with 127.0.0.1:56871
06.09.2016 - 12:52.59 || 127.0.0.1:56871 () -> Received "ID/-/eb72cd4a"
06.09.2016 - 12:52.59 || 127.0.0.1:56871 (eb72cd4a) -> Received "choose_crit_ABML/-/1"
06.09.2016 - 12:52.59 || No more data from 127.0.0.1:56871 (eb72cd4a)
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
06.09.2016 - 12:53.06 || Connected with 127.0.0.1:56874
06.09.2016 - 12:53.06 || 127.0.0.1:56874 () -> Received "ID/-/eb72cd4a"
06.09.2016 - 12:53.06 || 127.0.0.1:56874 (eb72cd4a) -> Received "choose_attr_ABML/-/{"1":{"attr":"30","k":"","sign":"<"}}"
['dej'] ['vel'] ['pri'] ['str'] ['str'] ['odp'] ['fin'] ['obr'] ['EBI'] ['EBI'] ['cis'] ['sre'] ['kap'] ['fin'] ['den'] ['dlg']

06.09.2016 - 12:53.08 || No more data from 127.0.0.1:56874 (eb72cd4a)
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
06.09.2016 - 12:53.25 || Connected with 127.0.0.1:56876
06.09.2016 - 12:53.25 || 127.0.0.1:56876 () -> Received "ID/-/eb72cd4a"
06.09.2016 - 12:53.25 || 127.0.0.1:56876 (eb72cd4a) -> Received "choose_attr_ABML/-/{"1":{"attr":"28","k":"","sign":"<"}}"
['dej'] ['vel'] ['pri'] ['str'] ['str'] ['odp'] ['fin'] ['obr'] ['EBI'] ['EBI'] ['cis'] ['sre'] ['kap'] ['fin'] ['den'] ['dlg']

06.09.2016 - 12:53.27 || No more data from 127.0.0.1:56876 (eb72cd4a)
----------------------------------------------------------------------------------------------------
```

Slika 2.11: Izpis poteka izvajanja Python servisa na strežniku.

```php
<?php
    $data       = array();
    if(isset($_POST["sessionID"])){
        require_once("conf.php"); //naložimo lastno knjižnico za metode
        checkIn($socket, $_POST["sessionID"]);
        $podatki = getCriticalsABML($socket, "get_criticals_ABML");
        close($socket);
        $data["success"] = true;
        $data["podatki"] = $podatki;
        echo json_encode($data);
    }
    else{
        $data["success"] = false;
        die(json_encode($data));
    }
?>
```

Slika 2.12: Komunikacijska PHP skripta.

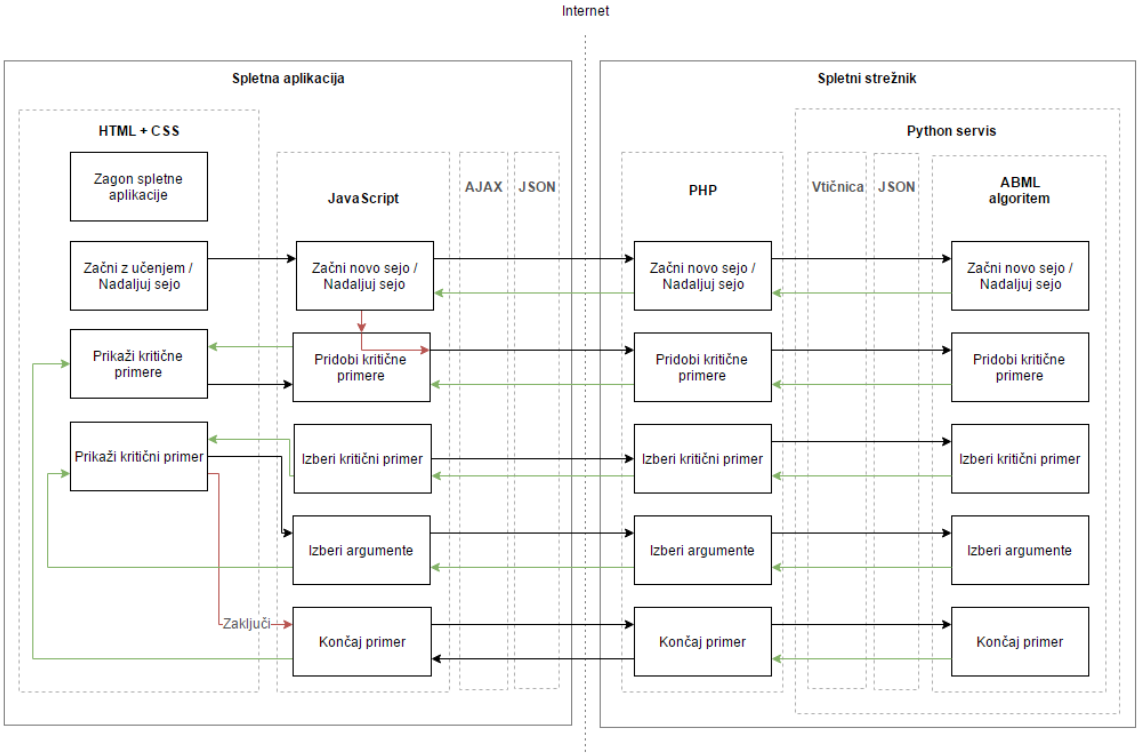
Kot je omenjeno tudi v podpoglavju 2.2.2, lahko pri gradnji PHP aplikacije uporabimo že narejena MVC (angl. *model-view-controller*) ogrodja (Slika 2.13). Takšna ogrodja temeljijo na večnivojskem generiranju spletne vsebine. Glavno vlogo prevzemajo kontrolerji, ki glede na uporabnikovo zahtevo kontrolirajo željen model v katerega vključijo posamezne poglede. Takšen sistem je zelo primeren za aplikacije z več podstranmi, za katere lahko izbiramo skupine in jim dodeljujemo grafične predloge. Ob morebitnih posodobitvah opravimo spremembo direktno na predlogi, to pa potem posodobi celotno strukturo prikaza na vseh podstraneh. Pri naši aplikaciji to ni potrebno. Zadostuje nam že klasično "ročno" generiranje izgleda strani brez vključevanja kompaktnih knjižnic. Poleg tega pa aplikacija temelji na enostranskem prikazu spletne strani (angl. *single page website*) [4]. Pri enostranskih izvedbah strani je pomembno, da je prikaz informacij dinamičen, zato pri realizaciji uporabljamo AJAX funkcije.



Slika 2.13: MVC struktura.

V sestavo sistema poleg strežniškega dela sodi tudi grafični aplikacijski del, ki se nato izvaja v odjemalčevem brskalniku. Pri tem je pomembno, da je struktura obeh delov sistema grajena skladno in je sinhrona pri izvajanju. Slika 2.14 prikazuje usklajenost funkcij glede na različne programske jezike in metode, ter kako med njimi poteka interakcija.

Izvajanje se začne z zagonom spletne aplikacije. Učenje se nato prične s klicom funkcije "Začni z učenjem oz. nadaljuj s sejo", ki se izvede z uporabo JavaScripta. Za komunikacijo nato uporabi AJAX tehnologijo. Podatke se pravilno strukturira v JSON notacijo in posreduje na spletni strežnik, kjer jih pričakuje PHP skripta. Od tam se informacije preko vtičnice prenesejo čez JSON dekoder, nato pa se začne procesiranje ABML-ja. Procesirani podatki se potem po isti poti vrnejo do spletne aplikacije. Takšno izvajanje se ponavlja ob vsaki na novo izvršeni akciji v spletni aplikaciji, v vrstnem



Slika 2.14: Delovanje in sestava celotnega sistema.

Poglavje 3

Rezultati

Poglavje vsebuje povzetek rezultatov. V aplikaciji je uporabljena domena, ki zajema podatke podjetij in njihovih bonitetnih ocen. Različica vmesnika je trenutno namenjena učenju in razumevanju teh podatkov. Opisi vseh uporabljenih značil oz. atributov so navedeni v [17].

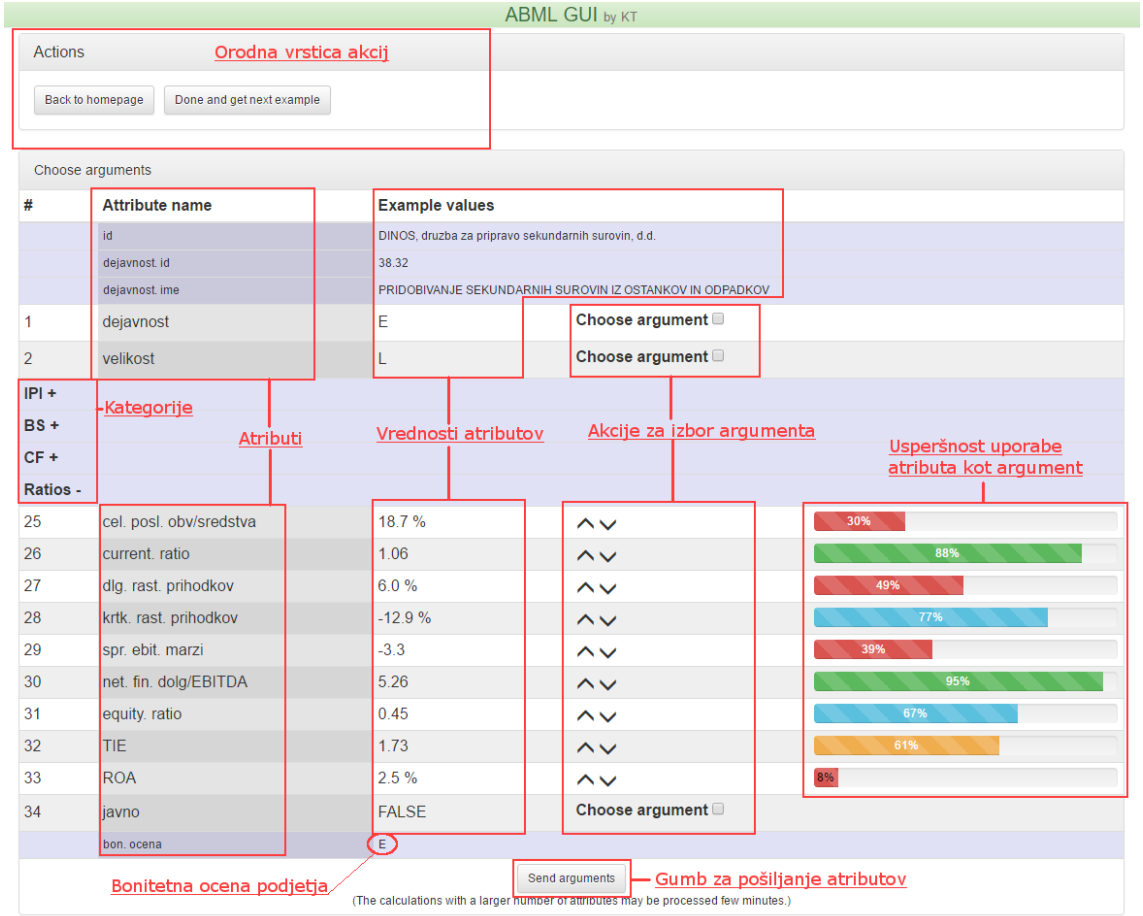
3.1 Grafični vmesnik in interakcija

Grafični vmesnik je oblikovan po standardih in načelih oblikovanja. Komponente so med seboj logično povezane in dajejo dinamičen interaktiven občutek. GUI za svoje delovanje potrebuje tudi strežniški del. Ta obsega Python servis in strežnik Apache, ki za pravilno izvajanje sistema potrebuje posebne PHP nastavitve. Primarno je uporaba celotne aplikacije namenjena za spletne strežnike, lahko pa ga poganjamo tudi lokalno.

Ob začetku nove seje se nam prikaže pet kritičnih primerov (Slika 3.1). V zgornji plošči vidimo nabor akcij. Možnost imamo tudi zapustiti program in ga nadaljevati kasneje s klikom na gumb "Nadaljuj prejšnjo sejo" (angl. *Resume previous session*) ali pa ustvariti novo sejo. Z izborom ene izmed petih vrstic nadaljujemo na naslednji prikaz, kjer lahko izbiramo atribute, s katerimi utemeljimo bonitetno oceno izbranega kritičnega primera (npr. bonitetna ocena je nizka, ker ima podjetje visok dolg in nizek dobiček).

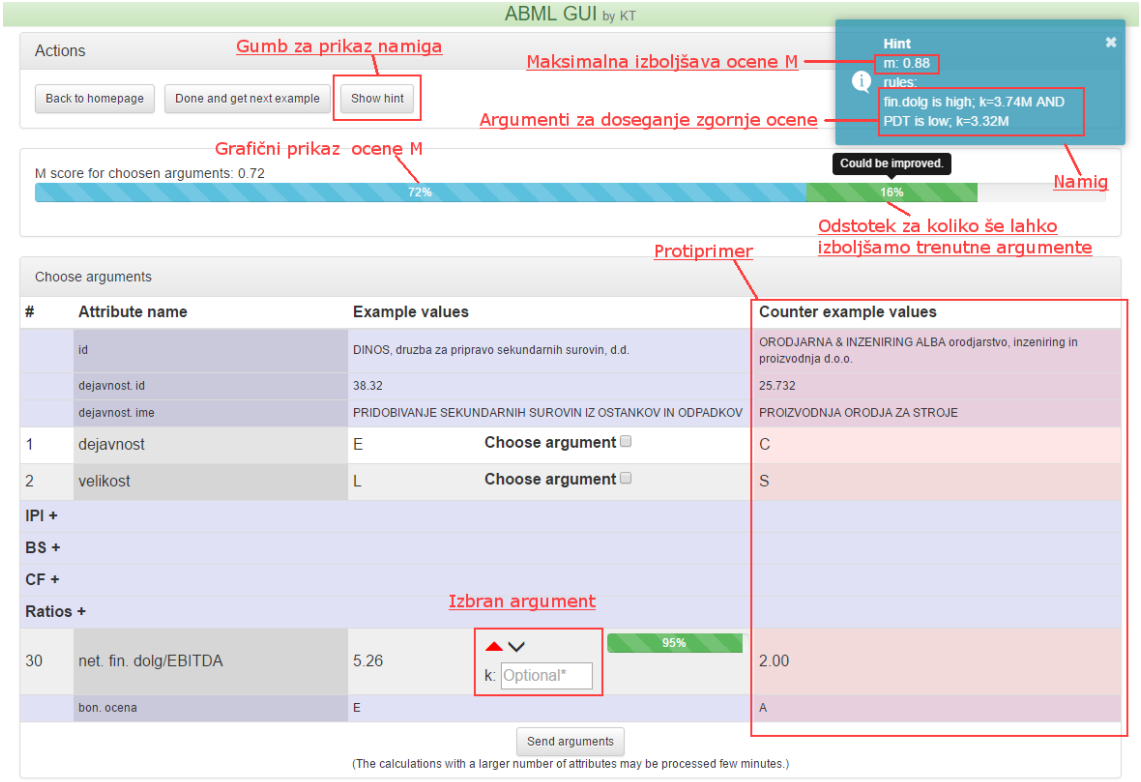
Slika 3.1: Izbiranje kritičnega primera.

Spodaj desno v skupini oz. kategoriji “Ratios” lahko vidimo grafični prikaz ocene vsakega izmed atributov, ki nam pove kako uspešni smo bili pri uporabi danih atributov v podanih argumentih (pri izračunu si pomagamo z M oceno). Ob slabših ocenah argumenta se prikazana vrednost pri atributih, ki so bili uporabljeni v argumentu, zmanjša in obratno, ob boljših ocenah se prikazana vrednost poviša (Slika 3.2).



Slika 3.2: Izbiranje argumentov.

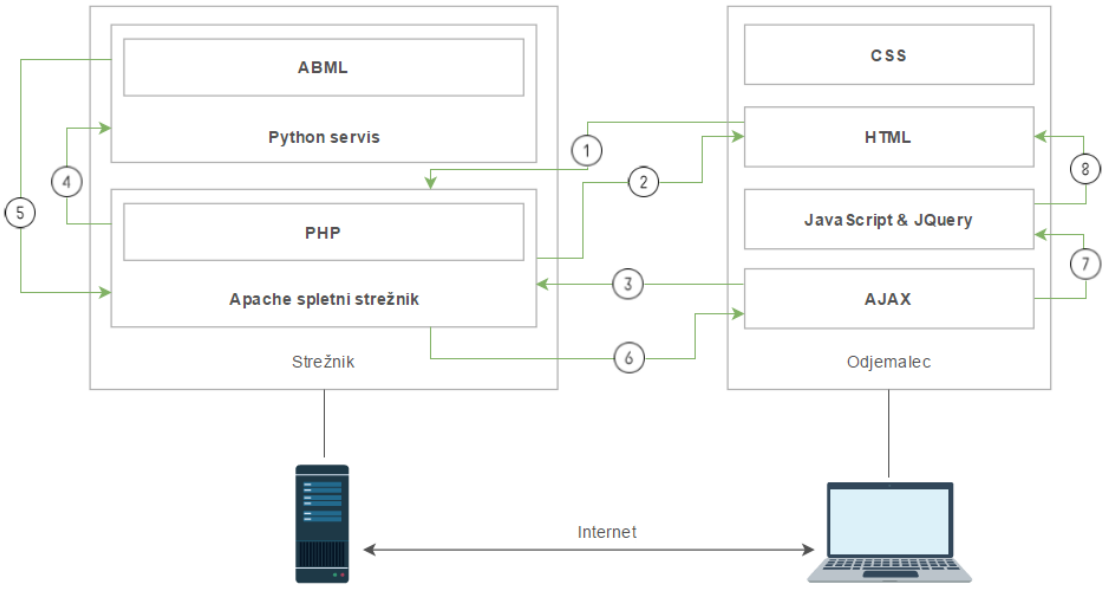
Po izbranih atributih in nastavljenih vrednostih argumente pošljemo strežniku. Na vrhu aplikacije se nam prikaže vizualizirana M ocena (Slika 3.3). Poleg ocene vidimo tudi delež, za koliko je še možno izboljšati oceno. Kot pomoč lahko s pritiskom na "Show hints" pridobimo tudi namige, kako lahko dosežemo višjo oceno. Na podlagi takšne interakcije se lahko človek ob primerih nauči branja domenskih podatkov. Seveda pa mora poznati pomene atributov, ki so opisani na osnovni strani aplikacije pod rubriko "*Help*" (pomoč).



Slika 3.3: Prikaz ocene M in protiprimera.

Komunikacija odjemalčevega brskalnika in strežniškega servisa deluje na podlagi več korakov (Slika 3.6). V prvem koraku odjemalec na strežnik pošlje zahtevo za dostop do aplikacije. Strežnik odgovori s podatki za prikaz aplikacije v HTML jeziku (korak 2). Odjemalčev brskalnik s pomočjo CSS pravil generira izgled grafičnega vmesnika in nato vseskozi čaka na uporabnikov nadaljnji ukaz (npr. klik na gumb), ki se potem s pomočjo AJAX funkcije posreduje spletnemu strežniku (korak 3). Ukaz se nato s pomočjo PHP skripte procesira in skupaj s podatki pošlje v Python servis (korak 4). Tam ga sprejme ABML algoritem, ki glede na želeno zahtevo pripravi podatke, jih pravilno strukturira in vrne nazaj spletnemu strežniku oz. PHP funkciji (korak 5). V šestem koraku se podatki iz strežnika pošljejo nazaj v odjemalčev brskalnik, kjer jih sprejme AJAX funkcija. Podatki se nato obdelajo

z uporabo JavaScript skripte, ki v kombinaciji z JQuery knjižnjico generira HTML prikaz (korak 7). Pri tem je pomembno omeniti tudi podatek zakaj se HTML generira na odjemalčevem delu in zakaj ne kar direktno na strežniku. Pojasnilo se nahaja v kontekstu z velikostjo prenosa podatkov od strežnika proti odjemalcu. Z uporabo JSON notacije velikost podatkov strnemo v karseda najmanjšo obliko. V primeru generiranja HTML kode na strežniku, bi to pomenilo obširnejšo količino podatkov, kar bi hkrati pomenilo tudi malo daljši odzivni čas. Zadnji, osmi korak pa zajema prikaz oblikovanih podatkov v brskalnik.



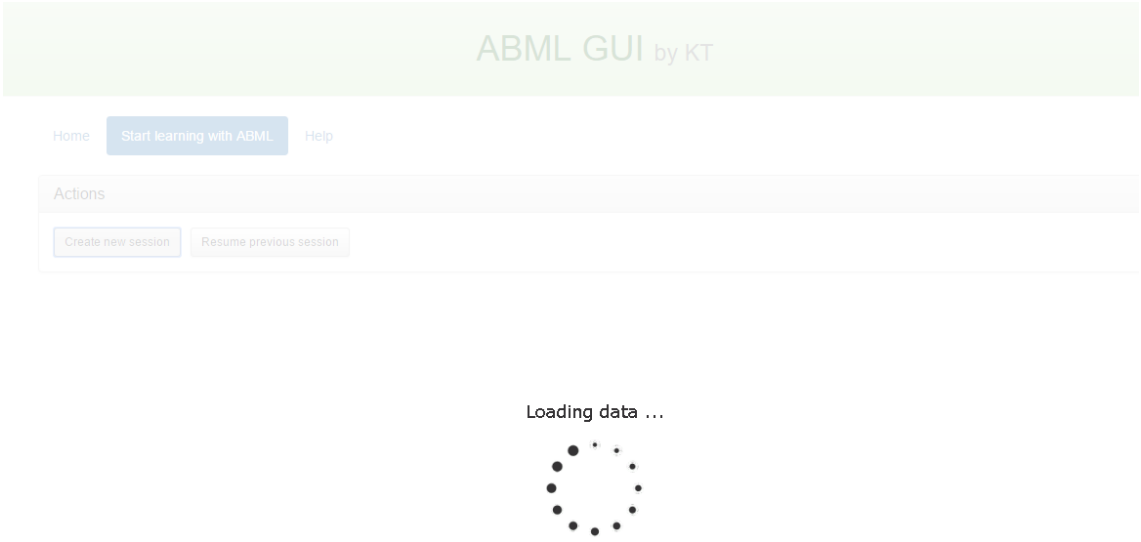
Slika 3.4: Shema komunikacije med odjemalcem in strežnikom.

3.2 Opis učne seje

Do sistema dostopamo z uporabo spletnega brskalnika. V brskalniku se odpre izgled strani v katerem izberemo podstran "Start learning with ABML" (Slika 3.5).

Slika 3.5: Spletna stran pred začetkom postopka učenja.

S klikom na akcijo "Create new session" bo aplikacija pognala strežniški del sistema. Med inicializacijo sistema se nam nato prikaže status nalaganja podatkov (Slika 3.6).



Slika 3.6: Zagon aplikacije.

Ko se nalaganje zaključi, se odpre pogled kritičnih primerov (Slika 3.7). Vsak kritični primer poleg vrstne številke prikazuje ime podjetja (ID), stopnjo odstopanja (Err) in bonitetno oceno podjetja (Class). Stopnja odstopanja

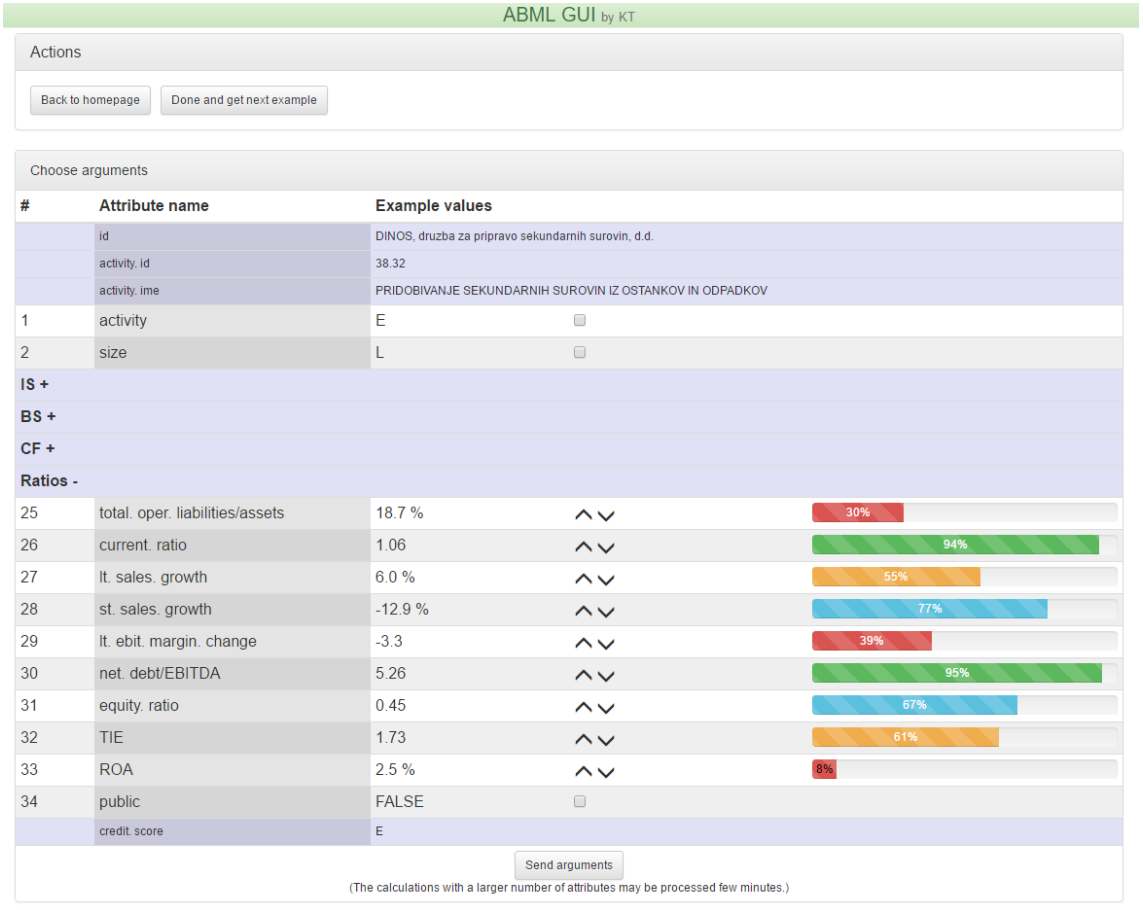
pomeni odstopanje kritičnega primera od drugih učnih primerov in se določi s postopkom v ABML sistemu. Ta nato za vsak učni primer vrne število v intervalu med 0 in 1, ki nam predstavlja stopnjo kritičnosti danega primera. Bonitetno oceno je vsem primerom v učni množici predhodno določil domenski strokovnjak.

Slika 3.7: Izpis kritičnih primerov.

S klikom na izbrani kritični primer odpremo pogled vseh značilk oz. atributov podjetja (Slika 3.8). V našem primeru smo izbrali prvi kritični primer. Tabela na čisto levi strani prikazuje vrstno številko atributa, ki ga lahko izberemo kot argument. V desni smeri nato sledi ime atributa in vrednost izražena bodisi s številom ali pa z odstotkom. Pri vrednostih opazimo tudi črne strešice s katerimi izražamo, ali je vrednost atributa previsoka ali pa prenizka. Poleg posebnih "Ratios" značilk je prikazan grafični prikaz statusa napredka atributa in predstavlja oceno, kako točni smo bili pri uporabi atributa. Če so naše napovedi z uporabo atributa pravilne, bo statusna vrstica prikazovala večji odstotek deleža.

Osnovne značilke so prikazane čisto na vrhu in jih ni mogoče skriti. Pod njimi se nahajajo glavni atributi in se lahko uporabljajo pri poučevanju.

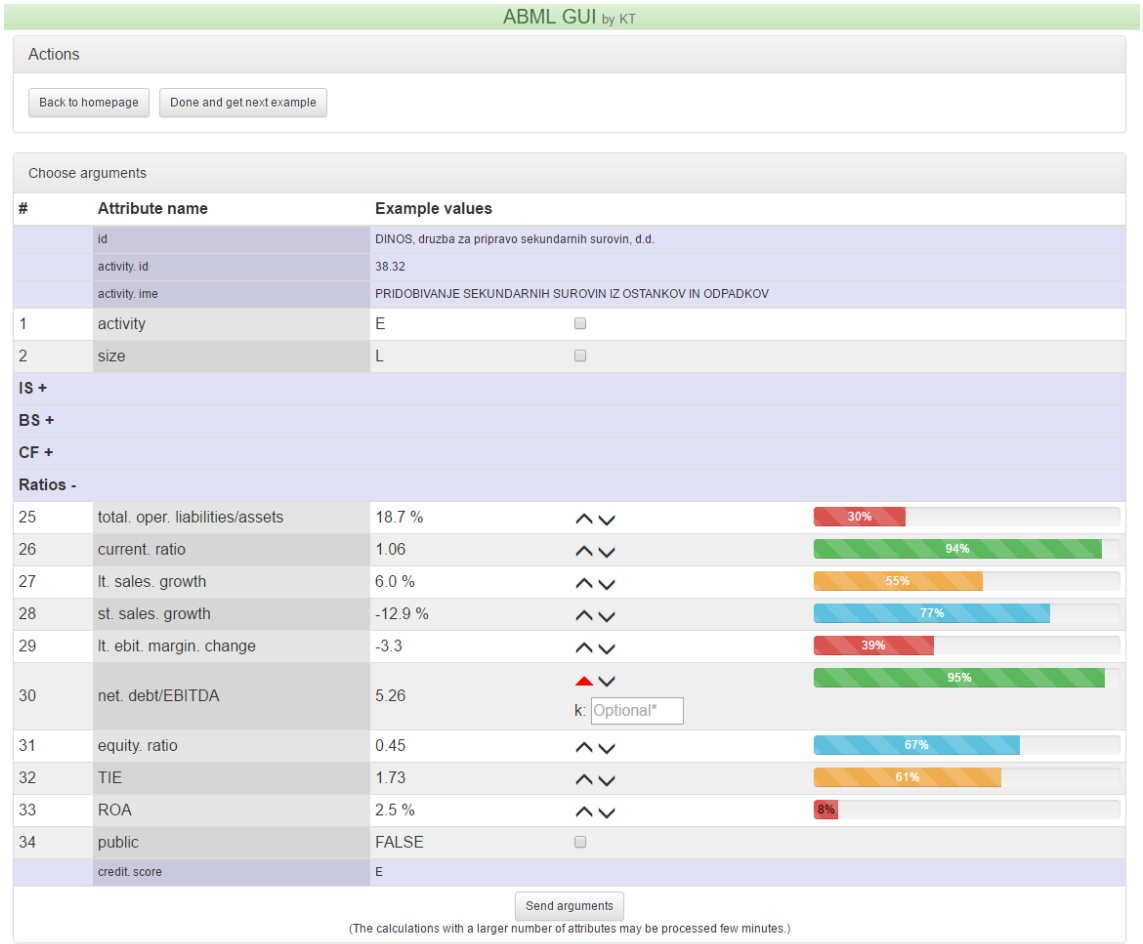
Zaradi velikega števila so razdeljeni v smiselne skupine oz. kategorije, ki so v začetnem prikazu skriti. Kategorije lahko nato poljubno skrijemo ali prikažemo, pri tem pa nam vedno ostajajo odprti tisti atributi, ki jih izbiramo za argumentiranje našega primera. V čisto spodnjem delu pogleda se nahaja predhodno omenjena kategorija "Ratios". Ta kategorija prikazuje posebne atribute, katere je dodal ekspert in ki lahko bistveno pripomorejo k lažjemu razumevanju učne domene. V našem primeru je kot razred podana bonitetna ocena podjetja in je vidna čisto spodaj v tabeli.



Slika 3.8: Prikaz podatkov kritičnega primera.

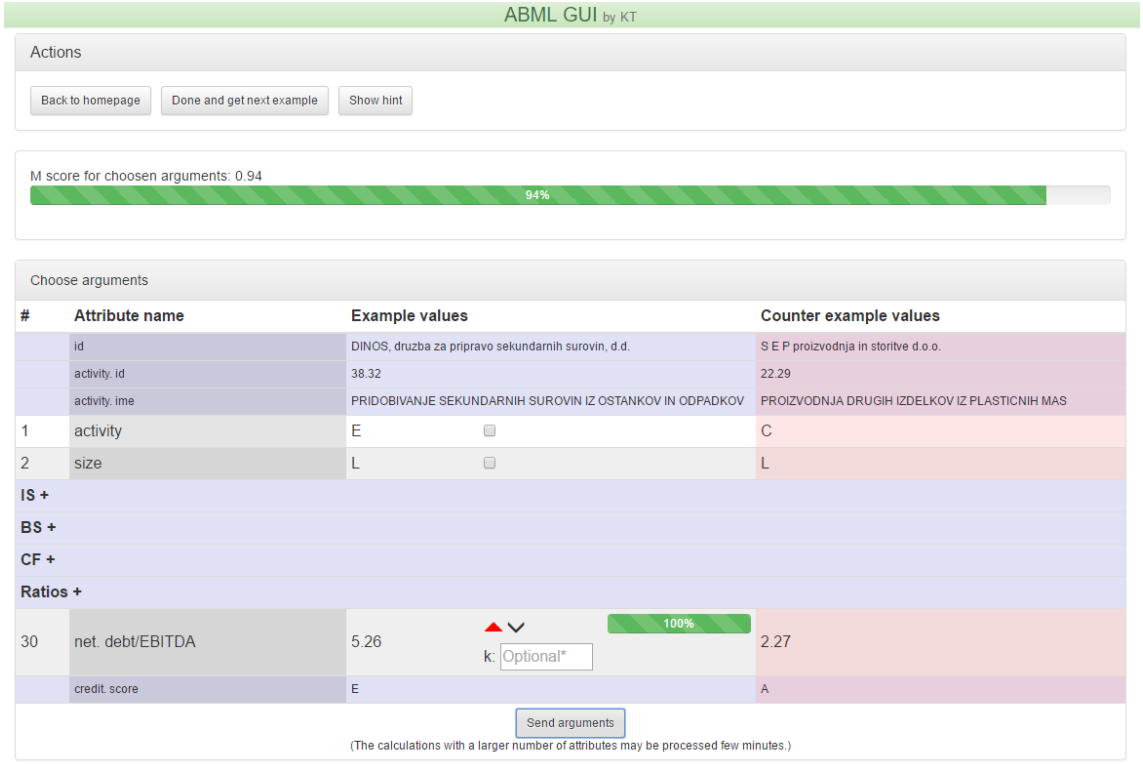
Po pregledu in razmisleku zakaj dani kritični primer spada v prikazani razred, izberemo izstopajočo značilko. Ob kliku želene izbire določimo, ali je

vrednost previsoka ali prenizka in se nam nato ustrezno označi izbran element (strešica se obarva *rdeče*), kot to prikazuje slika 3.9. Tukaj smo za atribut št. 30 navedli, da je njegova vrednost prevelika. Za tem se odpre tudi opcijsko okno, kjer lahko vnesemo primerno število k. Z njim lahko točno določimo mejo, do katere vrednosti bi bil atribut še sprejemljiv. Več napotkov, kako določati število k, se odpre, če se z miškinim kazalcem postavimo na vnosno polje. V tem primeru k pomeni, da bi se ocena podjetja lahko izboljšala, če bi se vrednost atributa nahajala v intervalu med številom k in trenutno vrednostjo atributa.



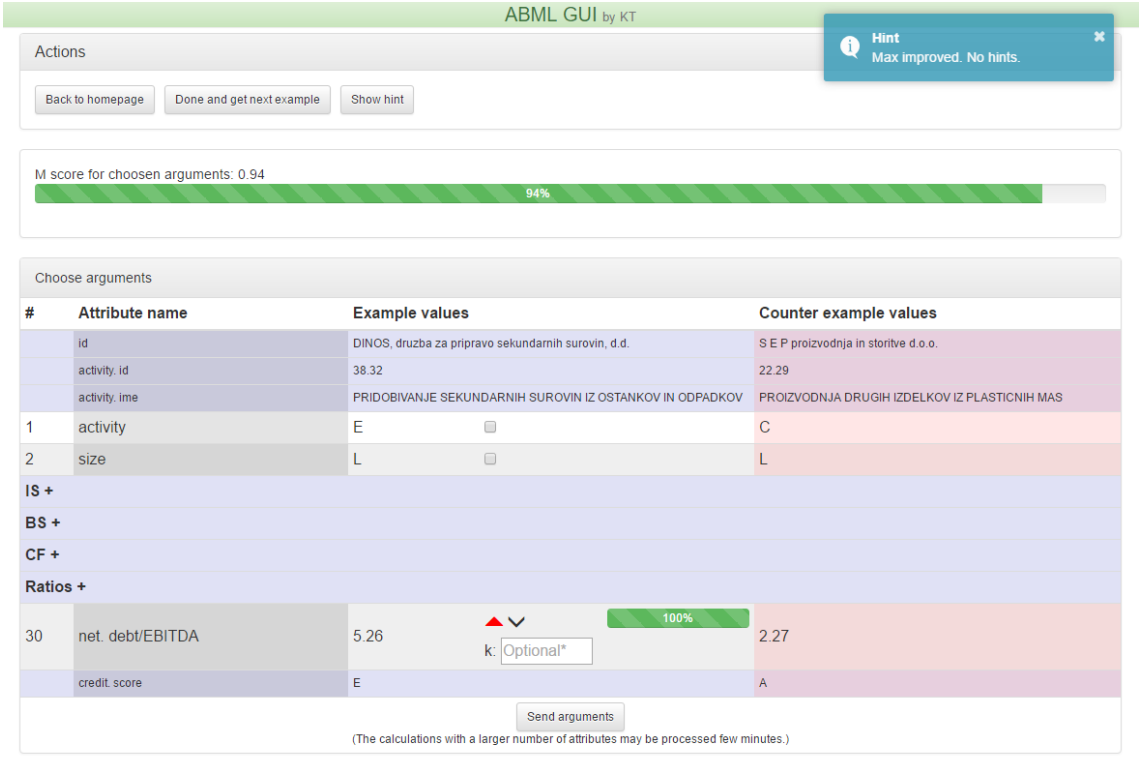
Slika 3.9: Izbor značilk za argumentiranje.

Po končanem izbiranju značilk s pritiskom na gumb "Send arguments", se samodejno skrijejo atributi, ki jih pri argumentiranju nismo izbrali. Odpre se nov pogled z dodatnimi informacijami (Slika 3.10). V zgornjem delu aplikacije se pojavi grafični prikaz ocene *M*, ki pove, za koliko smo z izbranimi argumenti izboljšali trenutni kritični primer. Poleg prikaza tabele kritičnega primera se na desni strani prikažejo podatki protiprimera glede na naš izbor značilk. S pregledom protiprimera lahko nato kritični primer dodatno argumentiramo in izboljšujemo. Pri tem se moramo zavedati, da lahko s prevelikim izborom značilk celotno zadevo popeljemo tudi v nasprotno smer. To pomeni, da se lahko ocena M spremeni tudi v slabšo. Da bi se temu izognili, se lahko držimo pravila in postavimo mejo ocene nad katero smo zadovoljni z argumenti. Grafični prikaz ocene *M* je odražen tudi z barvo. Uspešnost je označena z rdečo, rumeno, modro in z zeleno. Rdeča je v tem primeru najmanj uspešna, zelena pa najbolj. V primeru, da ocena ni najboljša in jo je mogoče še izboljšati, se pri grafičnem prikazu ocene M prikaže tudi delež, za koliko je mogoče izboljšati primer. Delež, ki ga je mogoče izboljšati se ustrezno označi z različno barvo in se nanj napiše odstotek izboljšanja, ki ga je mogoče doseči z izboljšavami. Primer trenutne slike prikazuje zelo dobro oceno M, kar pomeni, da smo primer dobro argumentirali. Spodaj v vrstici izbranega atributa, pa nam statusna vrstica uspešnosti uporabe argumenta, prikazuje 100 % dosežek. Kar nakazuje, da smo pri argumentiranju primerov pravilno uporabljali atribut *net.debt/EBITDA*. To lahko vzamemo tudi kot rezultat učenja, ki nakazuje razumevanje rabe te značilke.



Slika 3.10: Grafični prikaz M ocene in protiprimera.

Pri dodatnem argumentiranju so nam v pomoč tudi namigi. S klikom na akcijo "*Show hint*", se v zgornjem desnem kotu prikaže oblaček s sporočilom kako izboljšati argumente (Slika 3.11). Če izboljšave v takšni kombinaciji argumentov niso možne, potem to argument tudi sporoči. V nasprotnem primeru pa nam navede kolikšno oceno lahko dosežemo in kako do nje pridemo. V tem koraku vidimo, da našega izbora argumenta ni možno nadgraditi, saj smo že v prejšnjem koraku dosegli maksimum izboljšave.



Slika 3.11: Prikaz namiga.

Ko smo z argumentiranjem zadovoljni in je ocena uspešnosti (ocena M) zadovoljiva, kritični primer zaključimo. To storimo z izborom akcije "*Done and get next example*". Aplikacija nato ponovno poišče kritične primere in jih prikaže v tabeli (Slika 3.12). V tej točki se lahko odločimo ali bomo nadaljevali z reševanjem primerov, ali pa bomo argumentiranje zaključili. Ob nadaljevanju lahko ustvarimo čisto novo sejo ali pa uporabimo prejšnjo.

Slika 3.12: Posodobljen prikaz kritičnih primerov.

Poglavje 4

Diskusija

Pomemben izziv pri implementaciji je predstavljala predvsem komunikacija med Python in PHP skriptami. PHP sicer podpira izvajanje Python skript. Vendar so slednje zmožne teči v neskončnost in hkrati izpisovati podatke, pri PHP-ju pa je nekoliko drugače. Izpisovanje podatkov se namreč prične šele po zaključenem procesiranju, tako da ob neskončni zanki podatkov ne bomo nikoli videli. Obstajala je rešitev, pri kateri bi celotno Matevževo kodo prepisali v manjše datoteke in bi vse skupaj kontrolirali preko PHP-ja, kar pa ni ravno dobra zamisel, saj je PHP veliko počasnejši. Poleg tega pa zaradi različnih sej (angl. *session*) ne moremo vzdrževati podatkov v RAM pomnilniku. Namesto RAM-a bi lahko uporabili bazo, kar pa bi celoten sistem občutno zakompliciralo in upočasnilo. Na koncu le ostanemo pri sklepu, da drugače kot z uporabo Pythona ne bo šlo, saj je bolj primeren za takšna procesiranja.

Do rešitve smo kasneje prišli z integracijo algoritma v servis. Z uporabo nitk (angl. *thread*) ustvarimo navidezne vzporedne poti dostopa do servisa in lahko nemoteno komuniciramo z več uporabniki hkrati. Opravljeni so bili testi dostopov v praksi, kjer se je implementacija izkazala za zelo uspešno. Veliko oviro je predstavljalo tudi malo dostopnih informacij glede takšnih problemov. Večinoma se takšne izvedbe uporabljajo predvsem s krajšimi/končnimi Python skriptami. Prav tako nismo našli ustrezne litera-

ture, ki bi opisovala problem večuporabniškega načina pri uporabi algoritmov argumentiranega strojnega učenja.

Izziv se je pojavil tudi pri izpisovanju vseh atributov iz učnih primerov. V našem primeru imamo več kot 30 atributov. Hkratno prikazovanje vseh podatkov kritičnega primera in protiprimera bi lahko pomenilo zmedo. Implementirati je bilo potrebno posebne funkcije, ki vse skupaj pravilno prikazujejo in je z njihovo pomočjo izgled podoben spustnim seznamom (angl. *dropdown lists*).

Poglavje 5

Zaključki

V projektu smo se poleg glavne tematike izdelave grafičnega vmesnika za na argumentiranju temelječ tutorski sistem, v veliki meri ukvarjali tudi s komunikacijo strežnika z grafičnim vmesnikom. Izziv je bil združiti delovanje dveh povsem drugačnih programskih jezikov in med njima prenašati večjo količino podatkov. Prenos le-teh mora biti zanesljiv in karseda hiter, saj strežniški servis procesira podatke v realnem času. Glede na hitro procesiranje je bilo potrebno primerno prilagoditi grafični vmesnik. Aplikacija je tako strukturirana kot enostranska stran, njena glavna prednost je, da se po klicu različnih akcij na strani posodabljajo le učni podatki. Ostala struktura strani vedno ostane ista in ni potrebe po vnovičnih prenosih vsebine iz strežnika.

Opravljena so bila testiranja glede uporabnosti, izgleda in kakovosti interakcije pri uporabi spletne aplikacije. Uporabnikom izbor barv ustreza, popravili bi le kakšne podrobnosti. Izboljšave bi naredili predvsem pri izpisu namigov in uvodnih navodilih, ki se kasneje ob normalni uporabi ne prikazujejo. Prikaz s spustnimi seznami se jim zdi popolnoma primeren. Opažajo, da je res veliko podatkov. Ideja bi bila, da se nekateri kompleksnejši podatki skrijejo in se prikažejo šele po določeni doseženi oceni.

Končni izdelek ustreza načelom in ciljem, ki smo si jih zadali pred začetkom izdelave. Še več, uspelo nam je ustvariti sistem, ki lahko poteka čisto ločeno od ABML-ja. Tako se procesiranje podatkov lahko odvija na enem strežniku,

na drugem pa skrbimo samo za spletno aplikacijo. To je uporabno predvsem, ker lahko postavimo več različnih vmesnikov, pri tem pa uporabljamo le en strežnik za procesiranje podatkov. Kot stranski produkt tega je nastal sistem za API klice. To pa nam v prihodnje omogoča enostavnejše razvijanje mobilnih in drugih aplikacij, ki so tako tudi neposredno povezane med seboj z uporabo istega strežnika. Pri tem velja omeniti tudi prednost sistema pri uporabi katerekoli podatkovne domene. Z ustrezno strukturo, ki bo na voljo v podstrani aplikacije z navodili za uporabo, bo možno naložiti popolnoma nove podatke s pomočjo katerih bi se radi učili. Trenutno se za učne primere uporablja domena bonitetnih ocen podjetij. Lahko pa bi denimo naložili podatke o klikih na določeni spletni strani, kombiniranimi s podatki prodaje, ki bi nam pomagali razumeti povezavo med številom klikov in rastjo prihodkov.

Celoten izdelek je razširljiv in omogoča veliko nadgradenj. V prihodnje bi lahko dodali uporabne statistike in izračune, ki bi še natančneje določili profil vsakega posameznika. S temi podatki bi pridobili zelo koristne informacije za namen poučevanja. Poleg tega, bi lahko z uporabo novih funkcij nadgradili tudi uporabniški vmesnik. Naknadno bi lahko izdelali dodatno platformo za eksperte in učitelje. S tem pa bi lahko ustvarili veliko kakovostnejši sistem, ki bi kasneje dejansko prešel v obširnejšo rabo aplikacije.

Literatura

[1] D Bauer and CR Cavonius. Improving the legibility of visual display units through contrast reversal. *Ergonomic aspects of visual display terminals*, pages 137–142, 1980.

[2] Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, 13(6):4–16, 1984.

[3] Scott Brandt, Gary Nutt, Toby Berk, and James Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 307–317. IEEE, 1998.

[4] Fu-Sheng Chiu. Single page website interface, February 27 2006. US Patent App. 11/362,014.

[5] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.

[6] Wilbert O Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.

[7] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[8] Vida Groznik, Matej Guid, Aleksander Sadikov, Martin Možina, Dejan Georgiev, Veronika Kragelj, Samo Ribarič, Zvezdan Pirtošek, and Ivan Bratko. Elicitation of neurological knowledge with argument-based machine learning. *Artificial Intelligence in Medicine*, 57(2):133–144, 2013.

[9] Matej Guid, Martin Možina, Vida Groznik, Dejan Georgiev, Aleksander Sadikov, Zvezdan Pirtošek, and Ivan Bratko. Abml knowledge refinement loop: A case study. In Li Chen, Alexander Felfernig, Jiming Liu, and ZbigniewW. Ra, editors, *Foundations of Intelligent Systems*, volume 7661 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2012.

[10] Jernej Kogovšek and Jurij Jaklič. *Vizualizacija informacij: diplomsko delo*. J. Kogovšek, 2009.

[11] Nathan Labhart, Dozent der Lehrveranstaltung, and Christian Doelker. Gui–oder?

[12] King-Hwa Lee, Robert Cram, and Anil Mukundan. Web client-server system and method for incompatible page markup and presentation languages, August 19 2003. US Patent 6,609,150.

[13] Kevin McArthur. *Pro PHP: Patterns, Frameworks, Testing and More*. Apress, 2008.

[14] Matej Možina, Matej Guid, Jana Krivec, Aleksander Sadikov, and Ivan Bratko. Fighting knowledge acquisition bottleneck with argument based machine learning. In *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 234–238. IOS Press, 2008.

[15] Martin Možina, Jure Žabkar, and Ivan Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10):922–937, 2007.

[16] Jakob Nielsen. 10 usability heuristics for user interface design. *Fremont: Nielsen Norman Group.[Consult. 20 maio 2014]. Disponível na Internet*, 1995.

[17] Matevž Pavlič. *Ocenjevanje kvalitete argumentov pri argumentiranem strojnem učenju*. PhD thesis, Univerza v Ljubljani, 2015.

[18] Sebastian Raschka. *Python Machine Learning*. Packt Publishing Ltd, 2015.

[19] Rswebsols. Php frameworks. Dosegljivo: `http://www.rswebsols.com/tutorials/programming/12-best-php-frameworks-2016`, aug 2016. [Dostopano 26.08.2016].

[20] Guido Van Rossum and Fred L Drake. *Python language reference manual*. Network Theory, 2003.

[21] Guido Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, 2007.

[22] Joel M Winett. Definition of a socket. Technical report, 1971.

[23] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 366–387. Springer-Verlag New York, Inc., 2008.

[24] Beverly Park Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann, 2010.

[25] Matej Zapušek, Martin Možina, Ivan Bratko, Jože Rugelj, and Matej Guid. Designing an interactive teaching tool with abml knowledge refinement loop. In *International Conference on Intelligent Tutoring Systems*, pages 575–582. Springer, 2014.

[26] Matej Zapušek, Martin Možina, Ivan Bratko, Jože Rugelj, and Matej Guid. Designing an interactive teaching tool with abml knowledge refinement loop (powerpoint slides). Dosegljivo: `https://ailab.si/matej/doc/Designing_an_Interactive_Teaching_Tool_with_ABML.ppsx`, jun 2014. [Dostopano 04.07.2016].

[27] Matej Zapušek, Martin Možina, Ivan Bratko, Jože Rugelj, and Matej Guid. Designing an interactive teaching tool with abml knowledge refinement loop. In Stefan Trausan-Matu, Elizabeth Kristy Boyer, Martha Crosby, and Kitty Panourgia, editors, *Intelligent Tutoring Systems*, volume 8474 of *Lecture Notes in Computer Science*, pages 575–582. Springer, 2014.